UNIVERSITAT POLITÈCNICA DE CATALUNYA
BARCELONATECH

Sistema de control para un interferómetro de determinación orbital

Tesis de Grado presentado a

Escola Tècnica d'Enginyeria de Telecomunicació de Barcelona

Universitat Politècnica de Catalunya

por

Otman Mahyou

En cumplimiento parcial

de los requisitos para la titulación en

Ingeniería de Sistemas Electrónicos

Tutor: Antoni Broquetas

Barcelona, enero 2018

Resumen

La presente memoria de proyecto abarca el diseño y la implantación de un sistema de control para un sistema de determinación orbital diseñado por el grupo *RSLAB* de la UPC llamado *GEOSAR*.

El sistema *GOESAR* está formado por múltiples receptores cada uno de los cuales incorpora múltiples subsistemas tales como osciladores sincronizados en fase, amplificadores, mezcladores, conversores analógico digital, fuentes de alimentación, etc. El objetivo del proyecto es realizar una supervisión completa y en tiempo real del estado de los subsistemas para detectar y corregir de manera ágil cualquier problema o anomalía.

El sistema de control está forma por tres microcontroladores que se coordinan y comunican con un computador monoplaca, que tiene por responsabilidad el control total del sistema. Para comunicar estos elementos se ha utilizado el bus estándar *SPI* implementado en C++.

Para facilitar la supervisión y el control del sistema, se ha diseñado una interfaz gráfica que indica el valor de los dispositivos, el estado y el número de errores detectados durante las últimas 24 horas, además se ha implementado un registro histórico de los errores del sistema. La interfaz gráfica se comunica con el computador monoplaca mediante el protocolo *TCP/IP.* Para su implementación, se ha utilizado el software libre QT; en el entorno *QtCreator*.

Como resultado de la implantación del sistema de control se consigue el control automático de los dispositivos electrónicos de *GEOSAR*, facilitando la tarea al grupo *RSLAB*.

Resum

La present memòria de projecte conté el disseny i la implantació d'un sistema de control per a un sistema de determinació orbital dissenyat pel grup *RSLAB* de la UPC anomenat *GEOSAR*.

El sistema GEOSAR està format per múltiples receptors cadascun dels quals incorpora múltiples subsistemes com ara oscil·ladors sincronitzats en fase, amplificadors, mescladors, convertidors analògic digital, fonts d'alimentació, etc. L'objectiu del projecte és realitzar una supervisió completa i en temps real de l'estat dels subsistemes per detectar i corregir de manera ágil qualsevol problema o anamalia.

El sistema de control està forma per tres microcontroladors que es coordinen i comuniquen amb un computador monoplaca, que té per responsabilitat el control total del sistema. Per comunicar aquests elements s'ha utilitzat el bus estàndard SPI implementat en C ++.

Per facilitar la supervisió i el control del sistema, s'ha dissenyat una interfície gràfica que indica el valor dels dispositius, l'estat i el nombre d'errors detectats durant les últimes 24 hores, a més s'ha implementat un registre històric dels errors del sistema. L'interfície gràfica es comunica amb el computador monoplaca mitjançant el protocol *TCP/IP*. Per a la seva implementació, s'ha utilitzat el programari lliure QT; en l'entorn *QtCreator*.

Com a resultat de la implantació del sistema de control s'aconsegueix el control automàtic dels dispositius electrònics de *GEOSAR*, facilitant la tasca al grup *RSLAB.*

Agradecimientos

En este apartado del proyecto me gustaría dar las gracias a aquellas personas que me han ayudado de una manera u otra a hacer este trabajo de fin de grado.

Primeramente, me gustaría agradecer a Antoni Broquetas y a Roger Fuster, por guiarme durante todo el proyecto, tanto en la realización de la parte práctica como en la redacción de la memoria; por su paciencia y las horas que han dedicado a ayudarme y solucionar cualquier imprevisto o problema que ha ido surgiendo a lo largo de este periodo.

Agradecer también a Marc Fernández su colaboración en el proyecto y los consejos proporcionados.

Por último, quiero agradecer a mi familia, por confiar siempre en mí y haber hecho de mí la persona que soy.

Finalmente, este trabajo ha sido financiado por el Plan Espanyol de Ciencia, Investigación e Innovación (MINECO) con los códigos de proyecto TIN2014-55413-C2-1-P y TEC2017-85244-C2-2-P. Unidad de Excelencia María de Maeztu MDM-2016-0600 financiada por la Agencia Estatal de Investigación, España.

Tabla de contenido

Lista de figuras

Lista de tablas

Glosario

RSLAB: *Remote Sensing Laboratory.* Centro de investigación UPC-ETSETB.

GEOSAR: *Geostationary Orbit Synthetic Aperture Radar.*

SAR: Synthetic Aperture Radar. Sistema orientado a la obtención de imágenes de alta resolución de la superficie de la tierra.

IGARSS: *International Geoscience and Remote Sensing Symposium.*

ARC: *Active Radar Calibrator.*

LNA: *Low Noise amplifier.* Amplificador electrónico utilizado para amplificar señales débiles.

Wilkinson: Divisor de potencia capaz de repartir la potencia de entrada entre un número m de salidas.

LVDS: *Low Voltage Differential Signaling. Sistema de transmisión de señales de alta velocidad.*

GUI: *Graphic User Interface.* Interfaz gráfica de usuario.

PLC: *Programmable Logic Controller.* Controlador lógico programable.

SCADA: *Supervisory Control And Data Acquisitio.* Software y hardware que permite controlar y supervisar procesos en tiempo real.

CPU: *Central Processing Unit.* Unidad central de procesamiento.

E/S: entradas/salidas.

RAM: *Random Access Memory.* Memoria de acceso aleatorio

ROM: *Read Only Memory.* Memoria de sólo lectura.

SBC: *Single-Board Computer.* Ordenador monoplaca.

IDE: *Integrated Development Enviroment.* Entorno integrado de desarrollo.

API: *Application Programming Interface.* Interfaz de programación de aplicaciones.

ANSI C: Estándar de programación para la portabilidad de código.

PCB: *Printed Circuit Board.* Circuito impreso.

UPD: *User Datagram Protocol.* Protocolo de comunicación unidireccional.

TCP: *Transmission Control Protocol.* Protocolo de comunicación bidireccional.

SPI: *Serial Peripheral Interface.* Estándar de comunicación.

ACK: *Acknowledgement.* Mensaje de confirmación de recepción de datos.

GNU: Sistema operativo formado por software libre.

Script: Archivo de órdenes.

SMD: *Surface Mount Technology.* Dispositivos de montaje superficial.

USB: *Universal Serial Bus.* Bus de comunicación para conectar y comunicar con dispositivos electrónicos.

TTL: *transistor-transistor logic.* Tecnología de construcción de circuitos electrónicos digitales.

FTDI: *Future Technology Devices International.* Conversor de USB a TTL.

1. Introducción

En este capítulo se explican las razones por las que se realiza este trabajo, así como los requisitos y objetivos a cumplir. Además, se hace un análisis del proyecto *GEOSAR* y se detalla la estructura de este documento.

1.1. Planteamiento del problema

El Grupo de Investigación en Teledetección de la UPC (*RSLAB*) está desarrollando un sistema de determinación orbital pionero para proporcionar la órbita de varios satélites geoestacionarios con una gran precisión. El sistema de monitorización orbital forma parte de un sistema de formación de imágenes de apertura sintética *GEOSAR* que aprovecha los satélites de comunicaciones geoestacionarios como transmisores de oportunidad. *GEOSAR* [1] permitirá monitorizar eventos con gran exactitud cuya naturaleza se desarrolle rápidamente como inundaciones, terremotos, volcanes y otros peligros naturales o provocados por el hombre.

A medida que el sistema aumenta en complejidad, se agudiza la necesidad de disponer de un sistema de control y alarmas que permita conocer el estado de los dispositivos electrónicos del sistema *GEOSAR*. Este sistema debe de ser capaz de interpretar diferentes nodos de información, tales como señales de alimentación, comunicaciones digitales y osciladores de referencia, interpretar su funcionamiento y comunicarse de forma remota con el software de control.

Por todo ello se ha decidido realizar un sistema autónomo de control, permitiendo al usuario consultar los datos y configurar el sistema a través de Ethernet [2] , mediante una interfaz amigable y de fácil uso. Esto implica la utilización de distintas tecnologías de diferentes áreas de la informática y electrónica: desde la programación, diseño e implementación de un sistema de hardware de control, hasta la creación de una interfaz gráfica.

1.2. Objetivos

El objetivo del proyecto es desarrollar un sistema autónomo de control de forma que sea capaz de funcionar durante largos periodos de tiempo, tenga gran escalabilidad, así como facilitar al usuario un total acceso tanto a la información como a la configuración.

La presente memoria abarca el diseño, la configuración, la programación y la implantación del sistema de control *GEOSAR*.

Para dar respuesta a las necesidades expuestas, se plantea el diseño del sistema de control, con dos niveles de diferenciación:

- Nivel de control: Compuesto por el hardware de control.
- Nivel de supervisión: Compuesto por el software de control.

El sistema de control a implementar pretende conseguir los siguientes objetivos:

- Identificar las condiciones de operación ideales y la mejor estructura de automatización y control para lograr el control óptimo del sistema *GEOSAR*.
- Implementar el software de control para la supervisión de *GEOSAR*.
- Integración compacta y estructurada del sistema de control.

1.3. GEOSAR

1.3.1. Inicios y actualidad

En 1978 el Dr. Kiyo Tomiyasu [3] propone un nuevo método para explotar la tecnología *SAR [4]* basado en posicionar satélites en órbita geoestacionaria. Aunque idealmente un cuerpo geoestacionario se mantiene inmóvil respecto a la tierra por definición, las diferentes perturbaciones orbitales (diversidad del campo gravitatorio terrestre, efectos gravitacionales de la luna y el sol ...) provocan que el satélite en cuestión se desplace en torno a su posición nominal haciendo una forma elíptica con una distancia de extremo a extremo de unos 100 km. Este desplazamiento podría ser explotado para sintetizar una apertura *SAR*. El estado de la técnica en 1978 era todavía pobre para afrontar los grandes retos de esta propuesta (déficit de señal, operación fina sobre satélites, etc).

En el año 2014, un conjunto potente de instituciones (ESA, UPC, Politecnico di Milano, SES ASTRA, Thales Alenia, etc) inician el proyecto *GeoSTARe* financiado por *ESA* realizando propuestas para la observación radar del continente Europeo desde un satélite Geoestacionario [5].

Este proyecto pretende estudiar la recuperación de la propuesta del Dr. Tomiyasu. Aún quedaban problemas por resolver así que el proyecto se estructura en distintos paquetes de trabajo, encargándose la UPC de la calibración terrestre del sistema. Entre otros, la calibración terrestre se encargará de determinar con la precisión que sea necesaria la posición del satélite, para poder enfocar las imágenes *SAR* de forma efectiva.

Inicialmente se determina que para enfocar una imagen del tamaño de toda Europa se necesitaría una precisión de un orden de magnitud inferior a la longitud de onda de la señal emitida por el satélite. En el caso más extremo, estamos hablando de una señal en la banda de los 12 GHz y por tanto la precisión necesaria es del orden de los milímetros. Esta precisión no tiene precedentes.

A partir de aquí la misión se divide en dos proyectos en paralelo. Por un lado, la UPC estudia de forma oficial la solución de calibradores terrestres activos (*ARC*). Se trata de una solución relativamente segura, pero con unos problemas logísticos evidentes (principalmente, que no se puede probar sin tener un satélite *GEOSAR* en órbita).

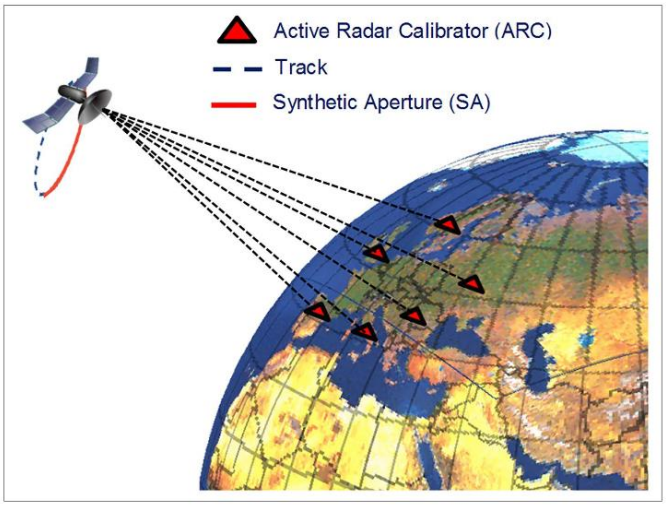

Ilustración 1.- Geometría básica del Sistema ARC con varios reflectores repartidos por Europa y el Satélite GEOSAR. Fuente: https://doi.org/10.5721/EuJRS20164938

Por otro lado, de forma no oficial, se estudia la opción de un interferómetro terrestre que se podría probar con cualquier satélite geoestacionario ya existente. Esta idea originalmente propuesta por el equipo de la UPC era muy arriesgada ya que a priori tiene muchas dificultades técnicas que no se sabe si se podrán resolver (sincronismo entre receptores, determinación orbital con datos interferométricas ...) así que se decide no hacerla pública hasta que se tengan resultados consistentes.

1.3.2. Funcionamiento básico del sistema.

La determinación orbital del sistema *GEOSAR* se basa en las técnicas de interferometría, cuyo principio se fundamenta en combinar señales generadas desde la misma fuente en diferentes lugares para obtener información muy precisa sobre la fuente.

El sistema hardware está compuesto por cuatro bloques:

- **Base del sistema:** Este bloque es el encargado de alimentar al sistema, proporcionar señales de reloj y procesar las señales interferométricas.
- **Interferómetro:** Es el encargado de combinar las señales de los satélites con el objetivo de obtener información útil sobre estos.
- **Calibración**: Este bloque se encarga de simular señales de satélites con el objetivo de establecer una referencia de control.

- **Monitoreo:** Este bloque se encarga de comprobar el correcto funcionamiento del sistema. Se diseñará e implementará en la presente memoria.

El funcionamiento del sistema hardware es el siguiente:

- La señal emitida por el satélite es recibida por N receptores del interferómetro. Cada receptor consta de una antena con un amplificador de entrada.

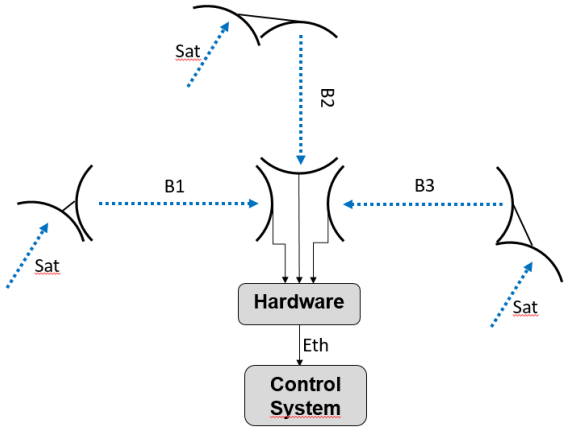

Ilustración 2.-Diagrama de bloques sistema GEOSAR. Fuente: Proporcionada por RSLAB-UPC.

- El interferómetro se basa en combinar en un correlador las señales recibidas desde antenas separadas a una distancia llamada línea de base. Para obtener una buena sensibilidad de la posición del satélite es conveniente separar las antenas del orden de 100 m. Para evitar las pérdidas, coste y problemas prácticos asociados con la utilización de cables, el sistema utiliza estaciones repetidoras en las ubicaciones más alejadas de las antenas, como se muestra en la Ilustración 2. De este modo la señal recibida por la antena se retransmite hacia el receptor central que realizar las correlaciones entre las diferentes señales recibidas.

- El sistema central consta de N antenas para recibir la señal de los N receptores.

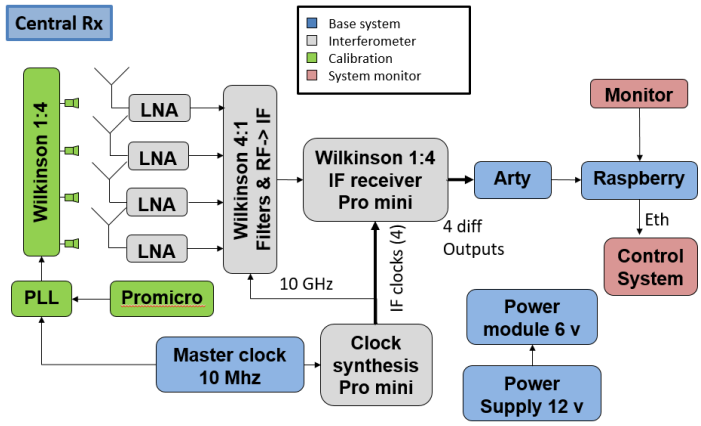

Ilustración 3.-Composición hardware GEOSAR. Fuente: Proporcionada por RSLAB-UPC.

- Todas las señales se amplifican y se combinan con un circuito de *Wilkinson* N: 1.
- Esta señal única resultante se filtra, se demodula a frecuencia intermedia(IF), se vuelve a filtrar y se amplifica.
- La señal se divide en M ramas con un divisor de *Wilkinson* 1: M. En este momento tenemos M señales, y cada una de estas señales es la suma de las N señales originales.
- Cada rama se amplifica, se demodula a una frecuencia diferente (dependiendo del satélite que queramos elegir) y se filtra. De forma que obtenemos M señales en banda base con un ancho de banda determinado por el último filtro (que es el más restrictivo).
- Cada rama en banda base se digitaliza a dos niveles según el estándar *LVDS* y se conecta a un puerto diferente de la *FPGA*.
- La *FPGA* está programada para "deshacer" la multiplexación de línea de base al tiempo que comprime la señal con varias correlaciones. En este punto tenemos tantas fases interferométricas como líneas de base tengamos multiplicado por el número de satélites detectados.
- Estas fases se envían a un servidor para su procesado a través de un computador monoplaca.
- El servidor de procesado se encarga de obtener la posición de los satélites a partir de las fases usando algoritmos de determinación orbital.

1.4. Especificaciones

Las especificaciones que se deben de cumplir son las siguientes:

- **Software**:
  - Implementación de un *front-end* para facilitar el control y la supervisión.
  - Visualización de los estados de los dispositivos en tiempo real, además de visualización de alarmas, avisos e históricos.
  - El aspecto visual de *GUI* ha de ser claro, compacto y estructurado para futuras modificaciones e incorporaciones.
  - Programación estructural y abierta para posibles modificaciones.
  - El set de instrucciones generado por la programación debe de ser intuitivo y replicable.
  - El software de control debe de comunicarse mediante Ethernet, de esta forma el control y la monitorización se podrá realizar desde un centro de control.
  - El software de control debe de poder ejecutarse en un PC con un entorno Windows.
- **Hardware**:
  - Diseño electrónico estructurado y replicable para futuras incorporaciones o modificaciones.
  - Capacidad para captar y procesar múltiples señales digitales y analógicas:
    - Control DC: Controlar la alimentación de seis dispositivos.
    - Control digital: Controlar señales digitales de siete dispositivos.
    - Control RF: Controlar señales de radiofrecuencia de seis dispositivos.
  - Capacidad de comunicarse con el software de control vía Ethernet.
  - El sistema debe de ser capaz de funcionar en un ambiente aislado.

1.5. Estructura de la memoria

La estructura de esta memoria es como sigue:

- En el capítulo 2 se expone el estado del arte donde se explica en qué consisten los sistemas de control y los distintitos tipos existentes. También se analiza el mercado de estas tecnologías comparando entre sí los distintos modelos que se pueden encontrar. Seguidamente, se explican, de manera breve, las tecnologías de desarrollo que se han utilizado en este proyecto.
- En el capítulo 3 se explica el diseño del sistema. Aquí se exponen los requisitos técnicos, los componentes utilizados, el diseño hardware de control, el diseño software de control, el protocolo diseñado para la comunicación entre hardware y software y la construcción del sistema de ensamblaje.
- En el capítulo 4 se exponen los resultados, donde detallan las pruebas realizadas para la validación del sistema.
- El capítulo 5 se expone los costes de los componentes utilizados, los costes de la creación del prototipo de control y los costes de licencias de software y otras herramientas.
- En el capítulo 6 se exponen las conclusiones, tanto del proceso de desarrollo como del estado final del proyecto, así como las posibles líneas futuras de éste.
- Finalmente, el capítulo 7 contiene la bibliografía utilizada.

2. Conceptos generales, tecnologías y herramientas.

En un sistema complejo donde la cantidad de dispositivos electrónicos es grande, es necesario disponer de una herramienta para controlar, automatizar y gestionar su funcionamiento. De esta manera se solucionan problemas como labores de revisión, mantenimiento, fiabilidad y productividad.

En nuestro caso, la implementación de un sistema de automatización y control permitirá que, mediante dispositivos de control (PLC's, SCADA, PC's,Raspbery,..), se pueda garantizar el funcionamiento del sistema de manera confiable. Además de esto, el uso de redes de comunicación permitirá que se conozca en tiempo real el nivel DC, las señales digitales y la frecuencia de operación de los dispositivos.

Con tal objetivo es necesario conocer cuestiones tales como: la tecnología que se utilizará, el modelo de automatización que se seguirá, el tipo de procesos que se tratarán, la red de comunicación que se implementará o el software de control que se implementará.

Con este propósito, en este capítulo se recoge información elemental sobre los modelos de sistemas de control más populares, las tecnologías de software de control, así como las herramientas utilizadas durante el desarrollo del proyecto.

2.1. Tecnologías de Control y Automatización

Para establecer un proceso de control y automatización es necesario hacer una correcta selección de equipos para integrar el sistema teniendo en cuenta diferentes factores como:

- Precio de acuerdo con su función (barato – caro, inseguro – seguro, desprotegido – protegido, austero – completo).
- Cantidad de entradas / salidas y si estas son analógicas o digitales y sus rangos de operación.
- La velocidad de procesamiento requerida según la aplicación.
- El nivel de tensión con el que se alimenta el dispositivo.
- Lenguajes de programación.

- La modularidad o capacidad de ampliación con la que cuenta el equipo.
- Capacidad y protocolos disponibles para establecer comunicación con otros dispositivos.

Las opciones tecnologías de automatización y control existentes pueden observarse en la siguiente tabla:

Tipo	Família Tecnológica	Subfamilias especificas	
Lógica Cableada	Eléctrica	Relés eclectromagénticos	
		Electroneumática	
	Electrónica	Electrohidráulica	
Lógica Programada	Electrónica	Sistemas Informáticos	Microcomputadoras
			Minicomputadoras
		Microsistemas	
		Autómatas Programables (PLC'S)	

Tabla 1.-Comparativa Sistemas de Control. Fuente: Elaboración propia.

El sistema de "lógica cableada" surgió como paso previo al de lógica programada. Estos sistemas cableados realizan una función de control fija, que depende de los componentes y de cómo están conectados entre sí. Son sistemas poco adaptables.

El éxito de los sistemas de “lógica programable” reside en el hecho de estar compuestos por elementos hardware comunes y lo que se cambia es el software. Esto permite una mayor adaptación a nuevas exigencias de la producción. Por otra parte, los equipos son relativamente baratos y no consumen grandes energías. En este proyecto se implementará un sistema de lógica programable. A continuación, se explica el funcionamiento de los principales controladores programables que se adaptan a los requisitos de nuestro proyecto.

2.1.1. Controlador Lógico Programable.

Un *PLC* [6] o autómata programable es una máquina electrónica programable capaz de ejecutar un programa, es decir, un conjunto de instrucciones organizadas de una forma adecuada para solventar un problema dado, y diseñada para trabajar en un entorno industrial y por tanto hostil. Las instrucciones disponibles para crear programas serán de una naturaleza tal que permitirán controlar procesos, por ejemplo: funciones lógicas,

operaciones aritméticas, de contaje de eventos, de temporización, etc. Además, el *PLC* estará diseñado de forma tal que la conexión del mismo con el proceso a controlar será rápida y sencilla por medio de entradas y salidas de tipo digital o analógicos. El siguiente diagrama muestra los componentes y la estructura de un *PLC*.

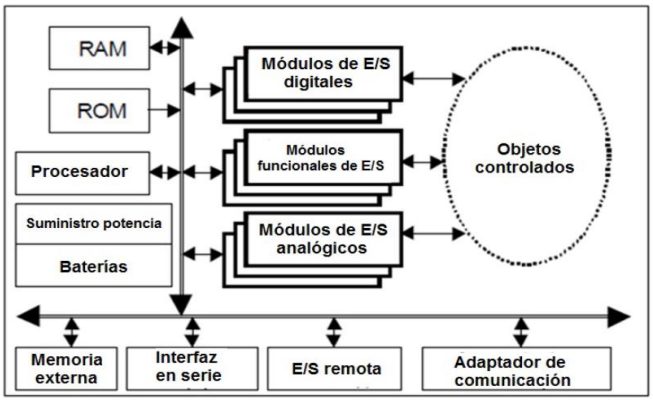

Ilustración 4.-Estructura General PLC. Fuente [5].

Como puede observarse en la ilustración 4, para que el sistema funcione es necesario que exista un suministro de potencia cuyo propósito principal es garantizar los voltajes de operación internos del controlador y sus bloques. Los valores más frecuentemente utilizados son ±5V, ±12V y ±24V y existen principalmente dos módulos de suministro de potencia: los que utilizan un voltaje de entra de la red de trabajo y los que utilizan suministradores de potencia operacionales para el control de los objetos.

La parte principal es la denominada "unidad central de procesamiento" o *CPU* que contiene la parte de procesamiento del controlador y está basada en un microprocesador que permite utilizar aritmética y operaciones lógicas para realizar diferentes funciones.

La transferencia de datos y direcciones en los *PLC's* es posible gracias a cuatro tipos de buses diferentes:

- Bus de datos: Es el encargado de la transferencia de datos entre la *CPU* y el resto de las unidades del sistema.
- Bus de direcciones: Es el encargado de indicar la posición de memoria o el dispositivo con el que se desea establecer comunicación.
- Bus de control: Es el encargado de enviar señales de arbitraje entre los dispositivos.
- Bus de sistema: Es el encargado de conectar los puertos con los módulos de E/S.

El lugar donde se guardan los datos y las instrucciones es la memoria que se divide en memoria permanente, PM, y memoria operacional, conocida como memoria de acceso aleatorio o RAM. La primera, la PM, se basa en las ROM, EPROM, EEPROM o FLASH; es donde se ejecuta el sistema de operación de la PLC y puede ser reemplazada. Sin embargo, la RAM, es donde se guarda y ejecuta el programa utilizado y no puede ser reemplazado [7].

Finalmente, los módulos de *E/S*, son aquellos de señal (SM) que coordinan la entrada y salida de las señales, con aquellas internas del PLC. Estas señales pueden ser digitales (DI, DO) y analógicas (AI, AO), y provienen o van a dispositivos como sensores, interruptores, actuadores, etc.

Durante los últimos años las ofertas de *PLC's* han crecido considerablemente, existen variedades para todo tipo de aplicaciones de control. A continuación, se hace una comparativa de los modelos que más se ajustan a nuestro proyecto, donde se resalta en color verde y naranja, qué especificación es mejor o peor respectivamente.

	Omron	Siemens	Industrial Shields	Mitsubishi	Allen Bradley
Modelo	CJ2M	SIMA S7-226	M-Duino	FX3GE	Micro 820
Ethernet	Si	Si	Si	Si	Si
Tarjeta SD	Si	Si	Si	Si	Si
Tensión de funcionamiento	5 V	5 V	5 V	5 V	5 V
Tensión de entrada	0-34 V	0-24 V	0-24 V	0-30 V	0-24 V
Puertos digitales E/S	14	20	4	24	14
Puertos analógicos de entrada	6	4	12	2	4
Bus de datos	SPI, I2C, RS2-232	SPI, I2C, RS2-23	SPI, I2C,RS2-232	SPI, I2C, RS2-232	SPI, I2C,RS2-232
Lenguaje de programación	XtraWare	Step Basic	Arduino/SoapBox	FX-Developer	ConnectComponend
Tipo de lenguaje	Ladder/Secuencial	Ladder/Bloques	Ladder/secuencial	Ladder/Bloques	Ladder /secuencial
Precio	350+IVA	280+IVA	275+IVA	325+IVA	285+IVA

Tabla 2.- Comparativa PLC comerciales. Fuente: Elaboración propia.

2.1.2. Microcontroladores

Un microcontrolador [8] es un circuito integrado programable, capaz de ejecutar ordenes grabadas en su memoria. Incluye en su interior las tres principales unidades funcionales de una computadora: Unidad central de procesamiento, memoria y periféricos de entrada y salida. En este apartado, se estudian las características más importantes de la plataforma más útil para nuestro proyecto, *Arduino.*

Arduino

Arduino es un microcontrolador basado tanto en hardware y software libre, es decir, tanto los componentes físicos como el código es libre, permitiendo el acceso a cualquier usuario de manera sencilla.

Las placas de esta plataforma se caracterizan, principalmente, por las posibilidades que ofrecen de interaccionar con el mundo real de una manera muy sencilla y amigable para el usuario. Permitiendo así que cualquier usuario, aún sin conocimientos informáticos o electrónicos, pueda utilizarlas y desarrollar sus propios proyectos.

La otra gran característica, es que permite la interconexión de módulos, denominados *shields*. Estos módulos sirven para añadir funcionalidades extra que nuestra placa no dispone. De esta manera podríamos añadir a nuestro dispositivo conexión de red, con el simple hecho de conectarle un módulo de red o Ethernet. Estos módulos se colocan directamente sobre la placa, utilizando los puertos de entrada y salida para comunicarse. Además, poseen entradas y salidas también, pudiendo utilizar varios módulos sobre una única placa, comprobando siempre la compatibilidad entre estos puertos.

Las ventajas de Arduino sobre otras plataformas son las siguientes:

- Coste reducido: otras plataformas tienen mayor coste y no ofrecen la versatilidad de Arduino.
- Entorno de desarrollo: el IDE de *Arduino* es muy sencillo, ofreciendo gran facilidad al usuario a la hora de crear proyectos y cargarlos en el dispositivo
- Multiplataforma: el software de desarrollo, o *IDE*, se puede ejecutar en Linux, Mac y Windows.

- Hardware y Software libre: tanto el hardware como el software utilizado por *Arduino* está disponible a la comunidad, posibilitando que usuarios avanzados puedan modificarlo si así lo deseen

Durante los últimos años, la oferta de placas de *Arduino* ha crecido considerablemente. A continuación, se hace una comparativa de los modelos más utilizados, donde se resalta en color verde y naranja, qué especificación es mejor o peor respectivamente para nuestro proyecto:

	Arduino UNO	Arduino Mega	Arduino Ethernet	Arduino Pro Mini
Microcontrolador	ATmega328	Atmega2560	ATmega328	ATmega328
Velocidad de reloj	16 MHz	16 MHz	16 MHz	8 MHZ
Memoria flash	32 KB	256 KB	32 KB	16 KB
Memoria Flash de arranque	0,5 KB	8 KB	0,5 KB	2 KB
SRAM	2 KB	8 KB	2 KB	1 KB
EEPROM	1 KB	4 KB	1 KB	512 bytes
Ethernet	No	NO	Si	NO
Tarjeta SD	NO	No	Si	NO
Tensión de funcionamiento	5 V	5V	5V	3.3 V
Tensión de entrada (recomendada)	7-12V	6-20V	7-12V	3.3-15V
Puertos digitales E/S	14	38	14	14
Puertos analógicos de entrada	6	16	6	6
Puertos digitales de E/S PWM	6	14	4	6
Bus de datos	SPI, I2C, UART, USB	SPI, I2C, UART, USB	SPI, I2C, UART, USB	SPI, I2C
CC por cada puerto de E/S	40 mA	40 mA	40 mA	40 mA
CC por puerto de 3.3V	50 mA	50 mA	50 mA	50 mA
Largo	68,6 mm	101,52 mm	68,6 mm	33mm
Ancho	53,4 mm	53,3 mm	53,4 mm	18 mm
Peso	25 gr	37 gr	25 gr	11 g
Precio	20 e+IVA	35 e+IVA	39,90 e+IVA	12 e+IVA

Tabla 3.- Comparativa Arduino comerciales. Fuente: Elaboración propia.

Los modelos *UNO* y *Ethernet* tienen las mismas especificaciones técnicas, con la diferencia de que el modelo Ethernet posee solo 4 puertos *PWM*, ya que algunos de ellos son usados para interaccionar con el puerto Ethernet. Además, incorpora una ranura de *SD*, característica muy importante, ya que permite la escritura y lectura de datos de manera externa.

Por otro lado, el modelo *Mega* tiene mayor capacidad de cómputo ya que tiene más memoria, permitiendo cargar programas más grandes y complejos. Además, incluye un mayor número de entradas y salidas, lo que permite interconectar más dispositivos a controlar.

Finalmente, el modelo *Pro Mini* [9], es una versión reducida del *Arduino Uno*, con algunas diferencias importantes: los márgenes de tensiones de entrada son más amplios, todas las entradas y salidas pueden utilizarse tanto forma digital como analógica y el precio es significativamente más reducido.

2.1.3. Microcomputadoras

Una microcomputadora o computadora de placa reducida (*SBC*) está formada por una única placa, donde se encuentran todos los componentes necesarios que forman un computador, ensamblados de forma directa y permanente.

Esto significa que el procesador, la memoria y el sistema de entrada y salida, así como otros posibles elementos, son inamovibles, aunque recientemente los fabricantes están permitiendo la interconexión de otros elementos mediante bahías y puertos de expansión.

Las características de los *SBC* ofrecen una capacidad de cómputo moderada, en un espacio muy reducido debido a la integración de sus componentes. Esto repercute directamente en el consumo eléctrico, permitiendo así su uso en infinidad de aplicaciones.

Los SBC son usados típicamente en entornos industriales o sistemas embebidos, donde los sistemas poseen unas especificaciones y necesidades muy concretas. Los SBC responden perfectamente a esta demanda, donde suelen formar parte de otros dispositivos de control o interacción. Además, gracias al auge del hardware y software libre, el uso de estas tecnologías para usos académicos se ha incrementado.

En este apartado, se estudian las características más importantes de la plataforma más desarrolladas hoy en día, *Raspberry-Pi.*

Raspberry-PI

Raspberry-Pi [10] es una plataforma del tamaño de una tarjeta de crédito, lanzada por la fundación caritativa *Raspberry Pi Foundation*, creada en 2009, y apoyada por el Laboratorio de Computación de la Universidad de Cambridge y por la compañía *BroadCom*.

Con estas placas se pueden realizar las tareas típicas de un ordenador personal, conectando un teclado y ratón estándar: navegar por internet, edición de documentos, etc. Por ello se pueden considerar perfectamente como miniordenadores. Además, tienen un precio inferior a 40€, por lo que su bajo coste las convierte en una opción muy interesante para nuestra aplicación.

La gran ventaja de esta plataforma es la variedad de sistemas operativos que se pueden utilizar, el número de en *E/S* y la versatilidad de los lenguajes de programación como C++ o Python.

A continuación, se hace una comparativa de los modelos existentes, donde se resalta en color verde y naranja, qué especificación es mejor o peor respectivamente para nuestro proyecto:

	Model A+	Model B+	Model B Gen.2
SoC	BMC2835	BMC2835	BCM2836
CPU	1176JZF-S 700 MHz	1176JZF-S 700 MHz	Cortex-A7 900MHz
RAM	256 MB	512 MB	1 GB
Memoria Flash de arranque	8 MB	8 MB	8 MB
SRAM	512 KB	512 KB	512 KB
EEPROM	11 KB	11 KB	11 KB
Ethernet	No	Si	Si
Tarjeta SD	SI	Si	Si
Tensión de funcionamiento	5 V	5V	5V
Tensión de entrada	3.3- 5 V	3.3-5V	3.3-5 V
Puertos digitales E/S	40	40	40
Puertos analógicos E/S	0	0	0
Puertos digitales PWM	2	2	2
Bus de datos	SPI, I2C, UART	SPI, I2C, UART	SPI, I2C, UART
CC por puerto de 3.3V	16 mA	16 mA	16 mA
Largo	65 mm	85 mm	85 mm
Ancho	56 mm	56 mm	56 mm
Peso	23 gr	45 gr	45 gr
Precio	29,95 e+IVA	34,95 e+IVA	39,95 e+IVA

Tabla 4.-Comparativa Raspberry Comerciales. Fuente: Elaboración propia.

2.1.4. Elección de sistema de automatización.

Los controladores lógicos programable poseen características similares: alta escalabilidad, acceso a red, número de entradas/salidas alto, precio alto, capacidad de cómputo elevado, etc. No obstante, el modelo *M-duino* basado en el microordenador *Arduino* resalta sobre los demás por su número de entradas analógicas, la capacidad de utilizar software libre y la facilidad de interconectar más dispositivos de control.

Teniendo en cuenta los requisitos básicos del proyecto, el acceso a red por parte de los dispositivos es obligatorio. Los dispositivos *UNO*, *Mega* y *Pro Mini* no poseen dicha conectividad, aunque se podría añadir acoplando un módulo Ethernet, pero esto aumentaría el coste del proyecto. Aunque el modelo *Ethernet* permite conectividad a la red, el número de entradas y salidas es muy limitado.

Los modelos analizados de la familia *Raspberry Pi* poseen características similares, precio bajo, tamaño reducido, número de entradas y salidas alto, compatibilidad con otros dispositivos, software libre, escalabilidad alta, etc. Estas características convierten a *Raspberry Pi* en una solución viable para nuestro proyecto. El único inconveniente de estos microordenadores reside en la falta de entradas analógicas.

El único dispositivo que cumple con nuestras especificaciones es el controlador lógico *M-duino* , sin embargo, se ha decidido utilizar alternativas más viables debido al precio y al tamaño físico de este dispositivo.

La solución adoptada consiste en combinar diferentes tecnologías, con el objetivo de beneficiarse de las características de cada uno y limitar las carencias de estos. De este modo, se ha decidido utilizar el dispositivo *Raspberry-Pi* y la plataforma *Arduino* por la compatibilidad y las características de estos dos componentes.

Se ha utilizado una *Raspberry-Pi mode B+* junto con *Arduinos Pro Mini.* Como se ha comentado anteriormente, las características de los dispositivos *Raspberry-Pi* son similares, así que se ha elegido este modelo por el precio y la conectividad vía Ethernet. En cambio, la elección del *Arduino Pro Mini,* se ha hecho por los siguientes motivos:

- Coste y tamaño reducido.
- Con el sistema propuesto, este dispositivo no tiene necesidad de disponer de mucha capacidad de cómputo.
- No es necesario disponer de conectividad.
- Dispone de un rango de tensión de entrada más amplio que otros dispositivos de la misma familia.
- Número de entradas analógicas alto.
- Número de entradas y salidas digitales alto.

Con esta estructura se obtiene un sistema económico, con una gran variedad de entradas y salidas, tanto analógicas como digitales, conexión vía Ethernet, escalabilidad y modularidad, así como facilidad para posibles modificaciones futuras.

2.2. Tecnologías de Desarrollo

En las tecnologías de desarrollo existen infinidad de paradigmas, lenguajes de programación y herramientas para desarrollar. Esas tecnologías se tienen que centrar en el tipo de sistemas que se va a desarrollar. La plataforma que se va a diseñar tiene distintas partes pero que de alguna manera se relacionan y se complementan entre ellas, por lo que se utilizarán diferentes tecnologías de desarrollo de distintas índoles.

2.2.1. Arduino

Como se ha dicho ya anteriormente, *Arduino* es una plataforma de microcontroladores, cuyo software y hardware son libres. Esta plataforma distribuye su propio *IDE* [10], el cual puede ser modificado por los usuarios.

Ilustración 5.- Logotipo Arduino. Fuente: https://www.arduino.cc/

El lenguaje de programación empleado es una mezcla de C y C++. Para facilitar el proceso de implementación, el *sketch* tiene predefinidas dos funciones. Una para la configuración

del dispositivo, denominada *Setup*, y otra de ejecución infinita denominada *loop*, donde iría el código principal de nuestro programa.

Desde el IDE se pueden realizar todas las tareas necesarias, tanto para crear programas como cargarlos en el dispositivo.

2.2.2. C++

C++ es un lenguaje muy versátil que se emplea en la programación de sistemas, como la construcción de intérpretes o compiladores, y en los últimos años ha ganado fuerza como herramienta para el desarrollo de aplicaciones. Este lenguaje está extensamente utilizado debido a varios motivos; entre ellos su estructura flexible y sencilla y la variedad de plataformas en las que puede ser compilado, así como el hecho de que tiene acceso a memoria de bajo nivel mediante punteros y que su aprendizaje no es complicado.

Ilustración 6.-Logotipo C++. Fuente: https://github.com/isocpp/logos.

El lenguaje de programación C ++ comparte muchas características con el lenguaje C, de hecho, como se puede deducir por su nombre, originalmente era una extensión de este último, llamada *C with clases*. Es por esto, que casi cualquier programa escrito en *ANSI C* puede ser compilado con un compilador C ++.

Resulta natural pensar pues que C ++ presenta algunas ventajas respecto C. Se trata de un lenguaje más complejo, pero tiene un rendimiento y seguridad superiores gracias a su continua evolución y optimización.

Se ha elegido este lenguaje para la programación de ***Raspberry-Pi***.

2.2.3. Qt

Qt [11] es un entorno de trabajo (*framework*) multiplataforma orientado a objetos que se usa principalmente para la creación de programas con interfaz gráfica de usuario (*GUI*); aunque también se utiliza para crear programas sin *GUI*, que integran diferentes tipos de herramientas para el terminal de pedidos. Por multiplataforma se entiende la propiedad de ser compilable sin aplicar cambios al código en varios sistemas operativos.

Ilustración 7.-Logotipo Qt. Fuente: https://www.qt.io/

Presenta un modelo de desarrollo de software libre y código abierto a través de *Qt Project*, que tiene un sistema de gobierno abierto con tanto desarrolladores individuales como empresas. Qt consiste en una serie de librerías escritas en lenguaje de programación C ++, que es su lenguaje de forma nativa.

2.2.3.1. Qt Creator

Para el diseño de la interfaz se ha utilizado *Qt Creator*, un entorno de desarrollo integrado (*integrated development environment,* IDE) también multiplataforma que utiliza las librerías de Qt. Utiliza un compilador C ++ de *GNU Compiler Collection Linux* e incluye un editor avanzado de código, que presenta varias características como autocompletado o resaltado de sintaxis, la herramienta de depurado visual (*visual debugger*) para C++ y *Qt Designer* que permite crear diálogos e interfaces gráficamente.

Dentro de *Qt Creator* se distinguen diferentes tipos de archivos según su función, distribuidos en carpetas que siguen la estructura siguiente:

- El archivo de proyecto de extensión .***pro***, que incluye a todos los demás.
    - *Header*: carpeta que contiene los archivos de extensión ***.h***, donde se declaran las funciones y variables.

- *Source*: carpeta que contiene los archivos de extensión ***.cpp***, donde se definen las funciones y variables.
- *Forms*: Carpeta que contiene los archivos de extensión ***.ui***, que se generan con *Qt Designer* y tienen formato de interfaz gráfica (ui).
- *Resource*: carpeta que contiene los archivos de extensión. ***qrc***, donde se almacenan archivos a los que el programa quiere tener acceso.

En la ilustración 8 se puede ver el aspecto de *Qt Creator*. Encuadrado en amarillo, en la zona donde se encuentran los proyectos, se observa la estructura mencionada con los diferentes archivos que forman el proyecto. El programa se encuentra en el modo de edición, por lo que se muestran los archivos en modo de escritura y lectura. Se observa que el texto se encuentra resaltado de un color u otro en función de su naturaleza.

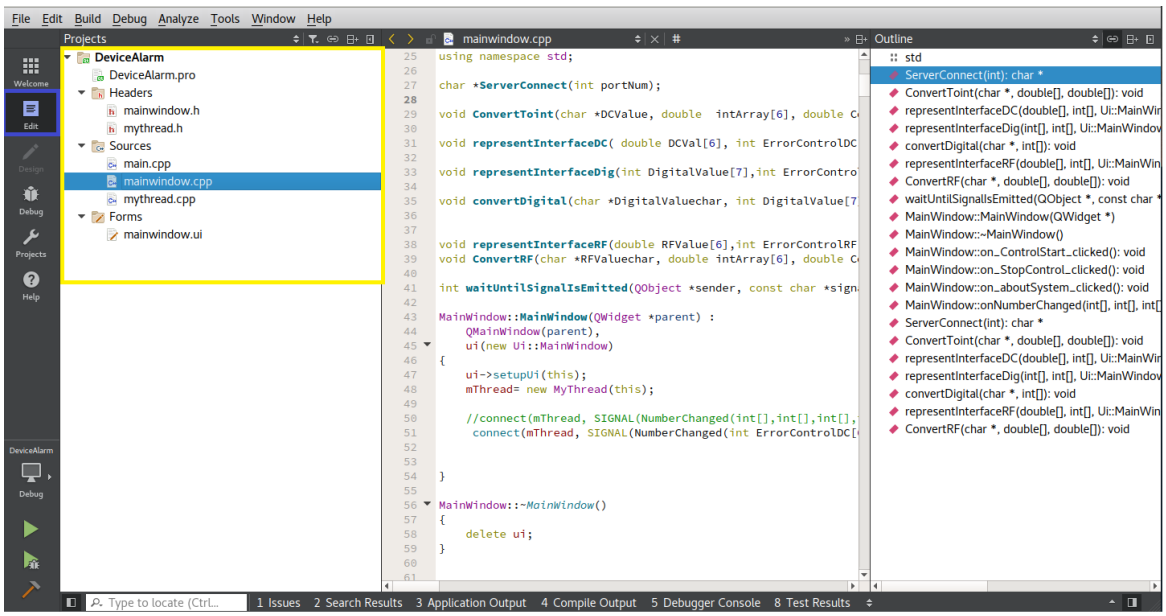

Ilustración 8.- Editor de código QT. Fuente: Elaboración propia.

En cambio, en la ilustración 9 , se muestra el modo de diseño de la *GUI* de *Qt Designer*. En este modo se aprecia el aspecto que tendrá la GUI cuando se ejecute el programa y se ofrecen varios widgets que se pueden insertar fácilmente arrastrándolos hasta la posición deseada. En la parte inferior de la figura y destacado en amarillo se encuentra el pulsador *Run*, que ejecuta el programa.

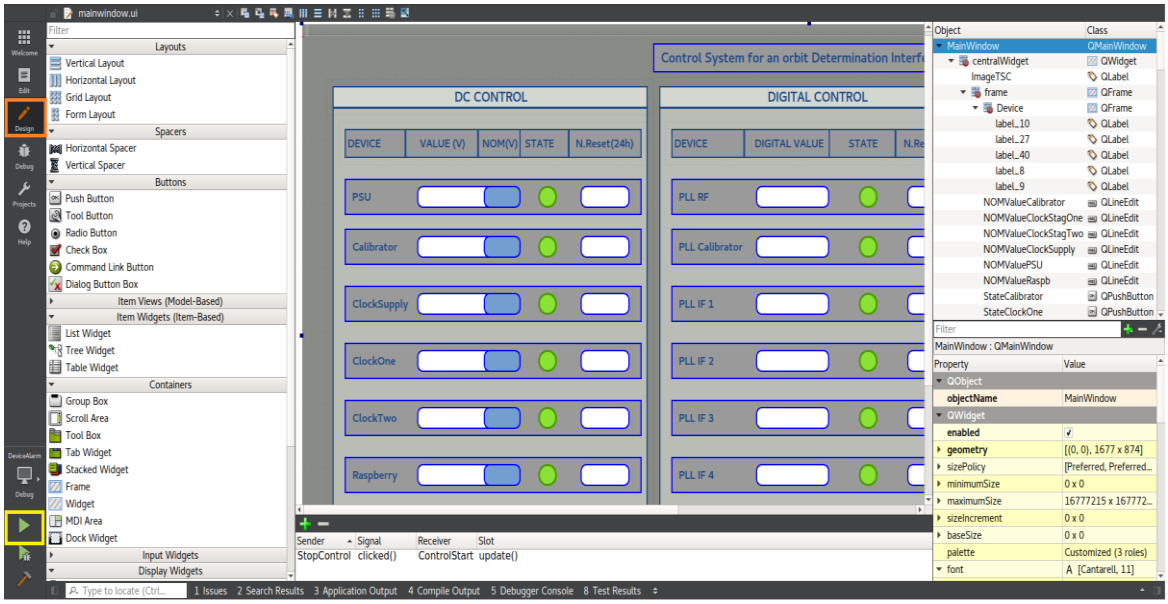

Ilustración 9.-Editor aspecto visual de la interfaz. Fuente: Elaboración propia.

2.2.3.2. Gestión de eventos con Qt

En este apartado, se explica, en términos generales, el funcionamiento del programa.

Una interfaz de usuario no es más que un medio de comunicación entre un usuario y una máquina o dispositivo. Mediante imágenes y objetos gráficos se crea una interacción entre estos dos elementos, que desencadena en una serie de eventos. Así pues, interesa conocer cómo se gestionan las acciones con sus reacciones deseadas.

Qt Creator distingue las acciones y las reacciones bajo los nombres de *signals* o señales y *slots*. Una señal se emite cuando sucede un evento en particular, cuando un objeto cambia de estado de alguna manera; mientras que un slot es una función que se llama en respuesta a una señal determinada. Los objetos de *Qt* que forman la *GUI* tienen señales y slots ya predefinidos, pero en algunos casos se crearán señales y slots propios que se adapten a las necesidades de la interfaz. Hay que tener en cuenta que los datos que genera la señal deben coincidir con las que recibe el slot.

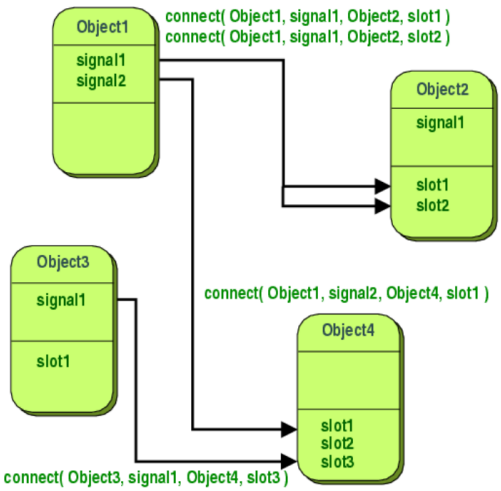

Ilustración 10.-Lógica de señales QtCreator Fuente: http://doc.qt.io/archives/qt-4.8/signalsandslots.html

Las señales y los *slots* se conectan mediante la función *connect*(), por lo que se pueden conectar varias señales a un mismo *slot* empleando diversas funciones *connect*() o viceversa. La función *connect*() se organiza siguiendo el procedimiento del ejemplo siguiente:

```
connect(ui->lineEdit,SIGNAL(textChanged(QString)),this,SLOT
(escr_text(QString)));
```

En el ejemplo se conecta el objeto *QLineEdit*, de nombre *lineEdit*, con el *slot escr_text ()* cuando el objeto detecta que ha cambiado su valor, que en este caso es de tipo texto. Además, la señal envía al slot el propio texto añadiendo *QString* a la señal y al *slot*. El texto resaltado en gris nos informa de donde se encuentran los elementos a enlazar, el *QLineEdit* se encuentra en la interfaz gráfica (*ui*), y mediante *this* informa que el slot se encuentra en la misma clase que la función *connect* ().

2.2.4. Eagle

Para el diseño de la placa de circuito impreso se ha utilizado Eagle (*Easy Applicable Graphical Layout Editor*) un entorno que permite dibujar diagramas esquemáticos de circuitos electrónicos, generar cada una de las caras de una placa de circuito impreso (*PCB*), así como la plantilla de perforaciones y las máscaras de soldadura utilizadas en la fabricación del PCB.

Ilustración 31.-Logotipo Eagle. Fuente: https://www.autodesk.com/products/eagle/overview

Entre la multitud de ventajas que ofrece este entorno, se ha empleado en este proyecto por los siguientes factores:

- Debido a su popularidad hay numerosas bibliotecas disponibles para descargar. Una biblioteca es un archivo que contiene uno o más componentes electrónicos o especificaciones de hardware traducidas al idioma de Eagle. Esto nos ha permitido utilizar componentes que no están en las bibliotecas predeterminadas.
- Permite generar nuevas librerías de componentes de forma sencilla.
- La versión Light, utilizada en este proyecto, está disponible de forma gratuita. Esta versión tiene algunas limitaciones: Generar PCB's con tamaño máximo de 100mm x 80 mm y solo permite generar circuitos impresos de dos caras.
- Facilidad de uso y familiaridad.

3. Diseño del sistema

En este capítulo se profundiza en el diseño del sistema de control, donde se explica el diseño hardware de control, el diseño software y los protocolos diseñados para la comunicación entre el sistema hardware y software.

3.1. Estructura y funcionamiento general del sistema

La estructura del sistema de control es la siguiente:

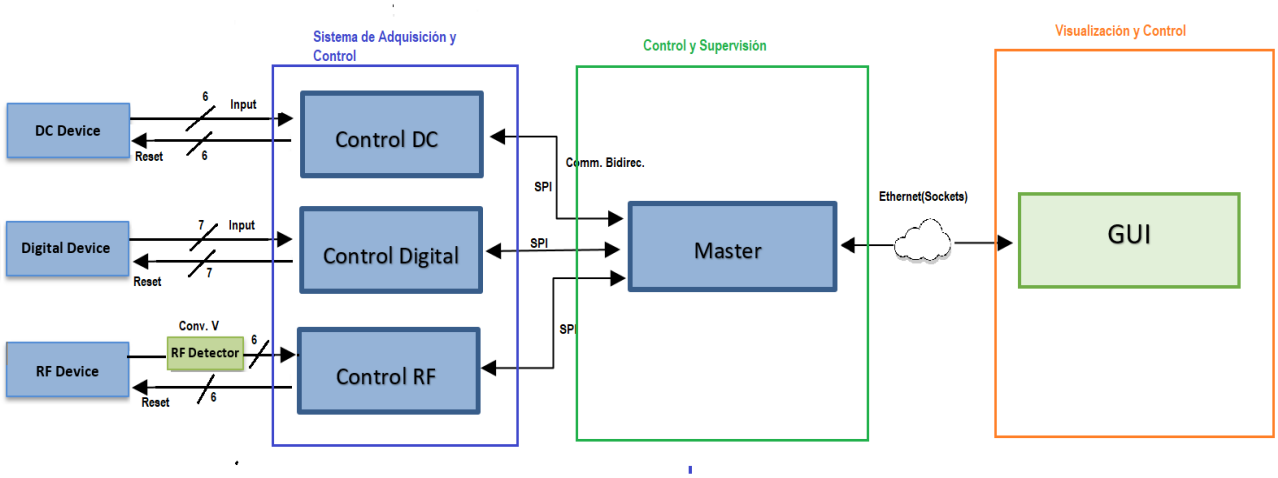

Ilustración 42.-Estructura Sistema de Control. Fuente: Elaboración propia.

El sistema se compone de los siguientes bloques:

- *Master:* Es el elemento central del sistema de control. Se encarga de las siguientes tareas:
  - Iniciar el sistema de control.
  - Iniciar la comunicación con los dispositivos de control.
  - Generar señales de reloj y control para la correcta transmisión de datos.
  - Recibir información de los dispositivos de control.
  - Transmitir la información de los dispositivos a controlar a la Interfaz gráfica.
- *Adquisición y Control*: Este bloque se encarga de adquirir, procesar y transmitir la información de los dispositivos a controlar. Está compuesto por tres elementos:
  - Control DC: Se encarga del control y la supervisión de las señales DC.

- Control Digital: Se encarga del control y la supervisión de las señales digitales.
- Control RF: Se encarga del control y supervisión de las señales frecuencia

- *GUI:* Este bloque se encarga de procesar y representar la información recibida en una interfaz gráfica para facilitar el control y la supervisión de los dispositivos.

En el diagrama siguiente se muestra el funcionamiento global del sistema:

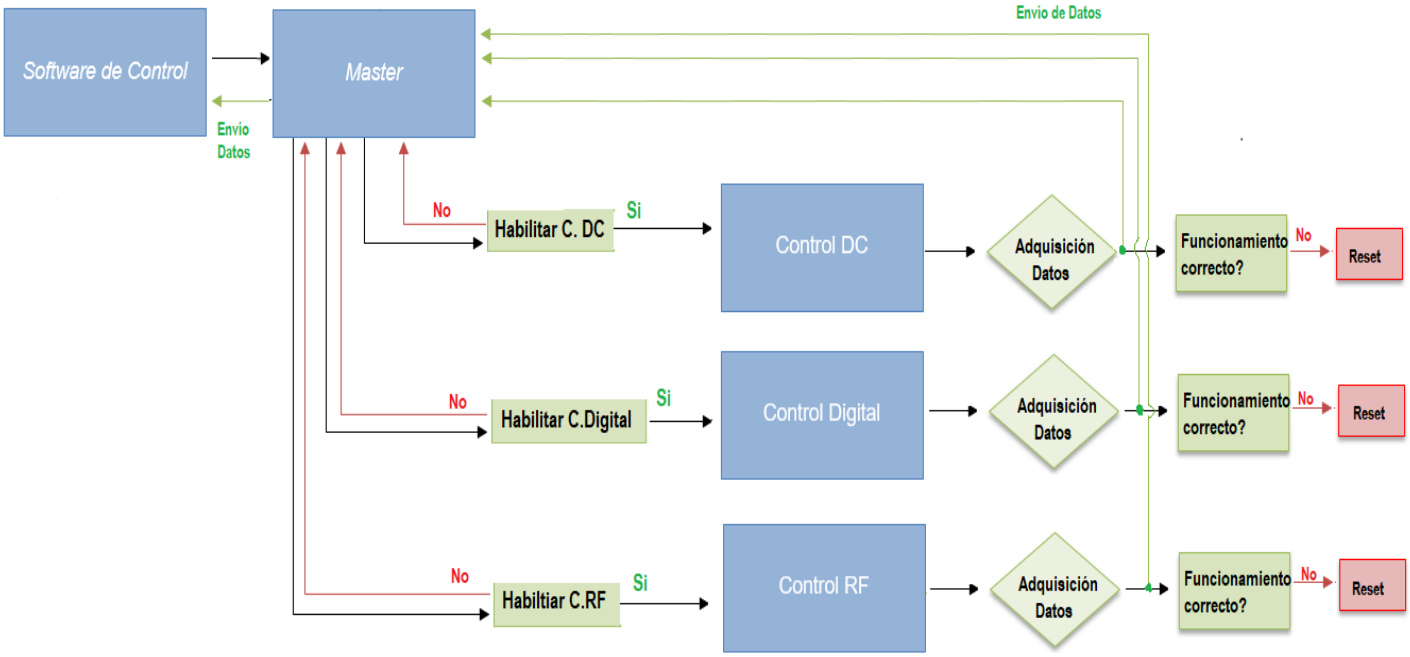

Ilustración 53.-Diagrama de funcionamiento general del Sistema de Control. Fuente: Elaboración propia.

- El usuario inicia el sistema de control a través de la interfaz gráfica. El software de control envía un mensaje al maestro indicando el inicio del proceso de control. El maestro devuelve un mensaje indicando la confirmación del mensaje y el inicio de la actividad.
- El maestro pone en marcha el sistema:
  - Envía un mensaje al controlador DC preguntando si está disponible y activo para la adquisición de datos. Si la respuesta es afirmativamente, el maestro permanece a la escucha hasta la recepción de datos.

- Cuando el Controlador DC envía el mensaje de confirmación, empieza el siguiente proceso de manera secuencial:
  - Adquisición de datos.
  - Envío de datos al maestro
  - Procesado de datos y *reset* de dispositivos en caso de mal funcionamiento.
- Cuando el maestro recibe los datos, se envían al software de control. Este bloque representa los datos, donde se indica el valor, el estado y el número de errores (por mal funcionamiento) de los dispositivos controlados.

- El maestro realiza el mismo proceso con el sistema de control digital y RF.
- El sistema de control permanece en bucle, hasta que el usuario finaliza el proceso.

En los siguientes apartados se describe el diseño y funcionamiento de cada bloque.

3.2. Diseño Hardware

En este apartado se describe el diseño del sistema hardware desarrollado, donde se detalla el funcionamiento, los componentes utilizados, la relación entre ellos y el proceso de integración. No obstante, antes de entrar en detalle, es necesario hacer una definición de los requisitos técnicos del sistema hardware *GEOSAR*.

3.2.1. Requisitos

Entradas Analógicas

La siguiente tabla muestra los dispositivos con entrada DC que han de ser controlados, donde se indica la etapa a controlar, los dispositivos que lo forman, la tolerancia de control y la función que realiza cada uno.

Etapa de Control	Abreviatura	Dispositivos	DC	Tolerancia de control	Función
Main Powe Supply	PSU	Fuente Regulada	12V	±3 %	Fuente de Alimentación Sistema GeoSAR
Calibrator Supply	CAS	Arduino Pro Mini	6V	±5 %	Proporciona señales de satélites de referencia.
Clock Synthesis Supply	CSS	Arduino Pro mini	6V	±5 %	Proporciona señales de sincronismo.
Stage One Supply	SOS	LNA+Wilkinson	6V	±5 %	Amplifican y combinan señales de satélites
Stage Two Supply	STS	FPGA	6V	±5 %	Desmultiplexación de señales
Servidor de red SBC	CAS	Raspberry Pi	6V	±5 %	Envío de señales a Servidor

Tabla 5.-Requisitos de control dispositivos DC. Fuente: Elaboración propia.

Entradas Digitales

La siguiente tabla muestra los dispositivos con entrada digital que han de ser controlados. En este caso no se define ninguna tolerancia, puesto que el dispositivo opera únicamente entre dos modos, valor lógico alto y valor lógico bajo.

Etapa de Control	Dispositivo electrónico	Abreviatura	Nº Elementos	Función
PLL Locked	PLL RF	PRF	1	Demodulación de señales de satélites a frecuencia intermedia
	PLL IF	RIF	4	Demoducion de frecuencia intermedia a banda base.
	PLL Calibrator	PCR	1	Demodulación de señales generadas a frecuencia intermedia
Stage Two	Arty FGPA	AFP	1	Desmultiplexación de señales

Tabla 6.-Requisitos de control dispositivos Digitales. Fuente: Elaboración propia.

Entradas Radiofrecuencia

La siguiente tabla muestra los elementos de radiofrecuencia que han de ser controlados.

Etapa de Control	Nº Elementos	Abreviatura	Frecuencia de operación	Rango de potencia	Comentario
PLL RF	1	PLRF	10 GHz	-10 dBm	Control de la frecuencia de operación para la correcta demodulación de las señales de satélites.
Satélite	4	ST1,ST2,ST3, ST4	0.7-2.7 GHz	-10 dBm	GeoSAR funciona con 4 satélites, en función de la frecuencia se escoge uno u otro.
RF Calibrator	1	RCA	10 GHz	-10 dBm	Control de la frecuencia de operación para la correcta generación de señales.

Tabla 7.-Requisitos de control dispositivos RF. Fuente: Elaboración propia.

Número de entradas/salidas del sistema

Finalmente, en la siguiente tabla se resumen el número total de entradas y salidas a controlar. Cada dispositivo, posee su respectiva entrada y se le asigna una salida para su control.

Tipo de Señal	Nº Entradas	Nº Salidas
DC	6	6
Digital	7	7
RF	6	6
Total	19	19

Tabla 8.-Número totales de entradas y salidas del Sistema de Control. Fuente: Elaboración propia.

3.2.2. Materiales utilizados

En este apartado se presentan y se explican los elementos que componen el sistema hardware de control.

LTC5533

LTC5533 [13] de *Linear Technology* es un detector de frecuencia *RF* de doble canal que proporciona un ancho de banda de entrada desde 300 MHz hasta 11 GHz. Esta característica, permite que el dispositivo sea conveniente para una amplia gama de aplicaciones *RF*. La potencia de entrada de este chip es de -32 dBm a 10 dBm. Como indica la curva de la ilustración 14, el chip genera una tensión de salida proporcional a la frecuencia *RF* de entrada.

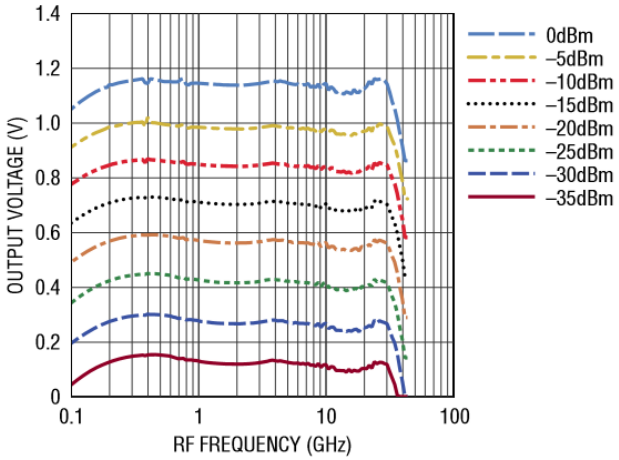

Ilustración 14.- Curva LTC5533 relación entre frecuencia de entrada y el voltaje de salida. Fuente: [13]

El chip *LTC5533* consume 30 mA y pueden funcionar con fuentes *VCC* de 3.3 V. También disponen de un modo de ahorro de energía que reduce el consumo de corriente a menos de 2 mA. En la imagen siguiente se puede observar el diagrama del chip y los componentes necesarios para su funcionamiento.

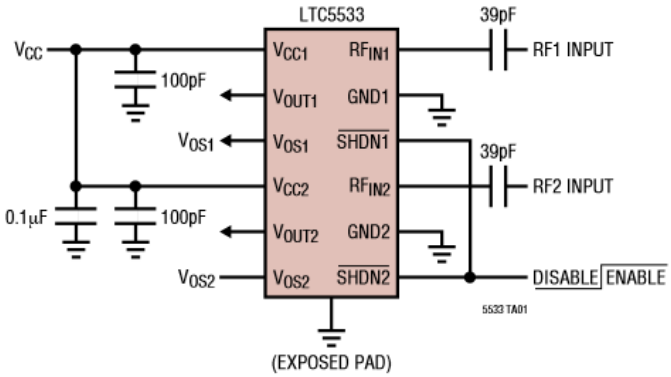

Ilustración 156.-Diagrama LTC5533. Fuente: [13]

Las características eléctricas más importantes de este detector de *RF* se muestran en la siguiente tabla.

Tipo de detector	Detector de picos *Schottky*
Frecuencia	300 MHz- 11 GHz
Potencia máxima de entrada	10 dBm
Potencia mínima de entrada	-32dBm
Pendiente logarítmica	29 mV/dB
Rango dinámico lineal	35 dB
Tensión de alimentación	3.3 V
Consumo	30mA
Dimensiones	4mm X 3mm

Tabla 9.-Características eléctricas LTC5533. Fuente: Fuente: Elaboración propia.

Como se indica en los requisitos, nuestro objetivo es detectar frecuencias de 10 GHz y 0.7-2.7 GHz con una potencia de entrada de -10 dBm. Con este objetivo, es importante conocer la relación que guardan estas frecuencias con el valor que proporciona el chip *LTC5533*. La siguiente tabla nos muestra estas relaciones.

Frecuencia (GHz)	Salida (mV)
10	833
2.7	878
0.7	881

Tabla 10.-Relación entre frecuencia y tensión de salida. Fuente: Elaboración propia.

Se deben de utilizar tres chips *TCL5533* para el control de RF.

Arduino Pro Mini

Arduino Pro Mini es un dispositivo con un tamaño relativamente pequeño, perteneciente a la familia *Arduino*, ideado para proyectos donde el tamaño es un factor limitante. Se ha elegido *Arduino Mini Pro* de 3.3V de 8 MHz debido a su pequeño tamaño, ya que no integra la electrónica USB y el microcontrolador es formato *SMD*, además tiene un consumo bajo y trabaja al mismo voltaje que *Raspberry-Pi.*

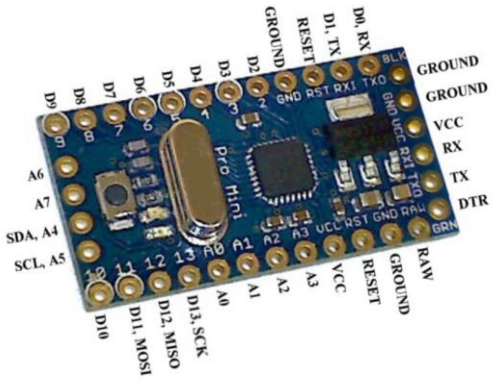

Ilustración 16.- Arduino Pro Mini 3.3 V 8 MHz. Fuente: https://www.tokopedia.com/carmelhill/arduino-pro-mini-atmega328-33v-8mhz

Las características más importantes de este dispositivo para nuestra aplicación son las siguiente:

- Posee 20 entradas, 4 sirven para comunicación *SPI* y 16 como pines de entrada/salida.
- El margen de tensión de entrada es de 3-15 V.

- La velocidad de reloj es del microcontrolador *ATmega328* es de 8 MHz.
- La corriente máxima de salida es de 50 mA.
- El consumo es de 13.5 mA.
- La dimensión es de 18x33 mm.

Esta placa se ha utilizado para controlar los dispositivos DC, Digital y RF. Su función consiste en adquirir datos, procesarlos y enviarlos al dispositivo central. Se han utilizado tres *Arduino Pro Mini*, una placa para cada controlador.

Raspberry Pi mode B+

Raspberry-Pi es una computadora de placa reducida de bajo coste y de bajo consumo empleada en proyectos de electrónica y robótica. El modelo utilizado en este proyecto, *Raspberry-Pi mode B+* [14], se trata de la primera generación de la familia *Raspberry-Pi.* Se ha elegido esta placa, por su compatibilidad con *Arduino,* la capacidad de comunicarse con otros dispositivos vía *SPI*, la capacidad de comunicarse vía ethernet, la capacidad de procesado de la que dispone y por el número de entradas y salidas que posee.

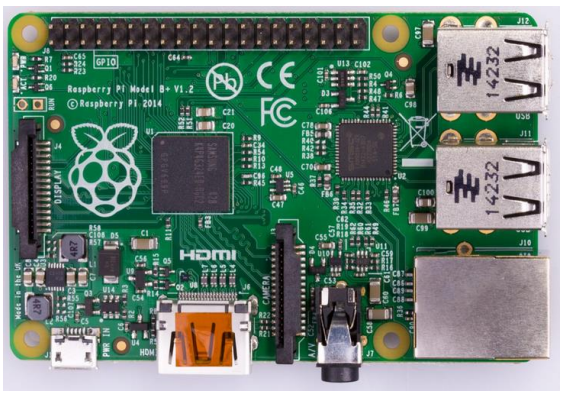

Ilustración 17.-Raspberry-Pi mode B+. Fuente: https://www.raspberrypi.org/products/raspberry-pi-1-model-b/

Las características más importantes de este dispositivo para nuestra aplicación son las siguiente:

- Posee un procesador *ARM 1176JZFS* a 700 MHz.
- Posee una memoria RAM 512 MB.

- *GPIO* (*General Purpose Input/Output*): Estos pines sirven de interfaz entre la *Raspberry-Pi* y el exterior. El modelo utilizado tiene 40 pines, 12 de ellos son de alimentación o tierra, 15 se pueden utilizar para establecer conexión con otros dispositivos y 13 de uso genérico.
- La corriente de salida es de 16 mA.
- Capacidad de comunicación vía *SPI* y ethernet.
- El consumo es de 900 mA.
- La dimensión de esta placa es de 85.60mm × 53.98mm

Esta placa constituye el elemento principal del sistema de control, su función es controlar los dispositivos *Arduino Pro Mini,* recibir información de éstos y transmitir esta información vía Ethernet a la interfaz gráfica. Se ha utilizado únicamente una placa.

CH340G

Entre las desventajas que presenta *Arduino Pro Mini* resalta la carencia de disponer de un conector *USB* para poder ser programado. Para solucionar este problema, es necesario disponer de un programador capaz de realizar la conversión del protocolo *USB* a *TTL Serial.* En este proyecto se ha elegido el programador *CH340G*.

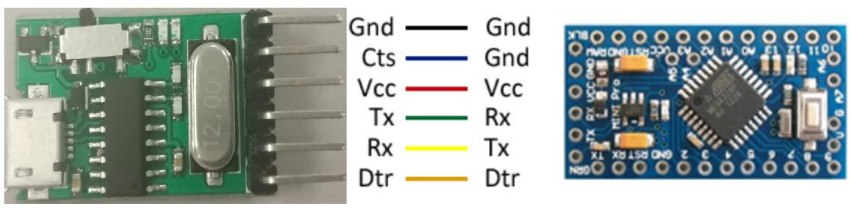

Ilustración 187.-Programador CH340G (izquierda) junto con Arduino Pro Mini (derecha). Fuente: Elaboración propia

En la siguiente tabla se muestran las conexiones necesarias para su funcionamiento:

Arduini Pro mini	CH340G
GND	GND
VCC	VCC
RXI	TX0
TDX	RXI
GRN	DTR

Tabla 51.- Conexiones Arduino Pro Mini y CH340G

3.2.3. Circuito sistema de control

En la figura siguiente se presenta el circuito diseñado para el control de dispositivos, donde se puede observar la relación entre los componentes utilizados y la estructura final del sistema de control.

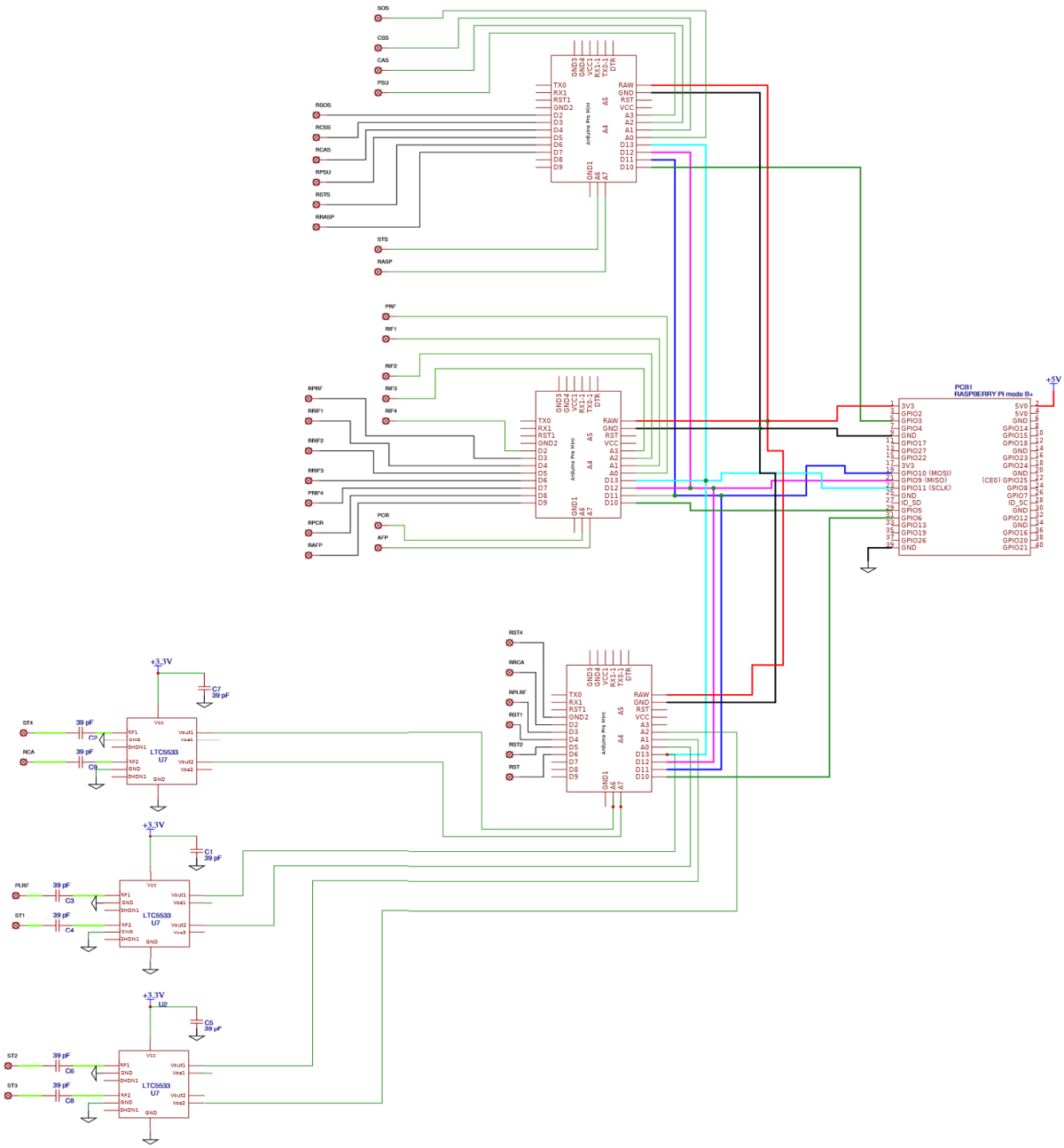

Ilustración 19.- Circuito Sistema de Control GEOAR. Fuente: Elaboración propia.

A continuación, se detalla la lógica de las conexiones y las características del circuito diseñado:

- Se utilizan tres dispositivos *Arduino Pro Mini* para el control DC, Digital y RF. Las líneas verdes indican las entradas de los dispositivos a controlar, mientras que las líneas negras indican las salidas *reset*.
- Se utiliza únicamente un dispositivo *Raspberyy-Pi mode B+* para el control de los dispositivos *Arduino Pro Mini.* Las líneas azul marino, azul turquesa y magenta indican respectivamente las entradas *MOSI, SCLK y MISO.*
- El dispositivo *Raspberry-Pi mode B+* es capaz de controlar únicamente dos dispositivos mediante *SPI* a través del bus *Chip Select (CE0 y CE1).* Nuestro sistema requiere del control de tres dispositivos, por tanto, se ha de buscar alguna alternativa o solución a este inconveniente. La solución propuesta, consiste en utilizar los pines *GPIO* de uso genérico para habilitar los dispositivos *Arduino Pro Mini.* Cada dispositivo *SPI* está asociado a un pin *GPIO* para la selección de esclavos. Cuando se realiza una transferencia, el pin *GPIO* asociado se pone a nivel lógico bajo, se realiza la transferencia y posteriormente se pone a nivel lógico alto. Este proceso se realiza automáticamente cuando se conecta el dispositivo que se quiere controlar al bus *CE.* El proceso descrito se puede realizar de manera manual con cualquier pin genérico *GPIO.* La solución hallada permite controlar hasta un número máximo de 13 dispositivos, otorgando al sistema mayor modularidad y facilidad de integración.
- Se deben de utilizar tres dispositivos *LTC5533* para la detección de frecuencia.
- Todos los dispositivos se alimentan a 3.3 V exceptuando *Raspberry-Pi* que necesita una fuente de alimentación de 5V.

En la siguiente tabla queda reflejado, a modo de resumen, los pines de salida y entradas de los dispositivos utilizados.

Raspberry Pi Mode b+	
Elemento conectado	Pin
MISO	21(GPIO11)
MOSI	19(GPIO10)
SCLK	23(GPIO11)
Arduino Pro Mini DC	5(GPIO3)
Arduino Pro Mini Digital	29(GPIO5)
Arduino Pro Mini RF	31(GPIO6)
Arduino Pro Mini	
Elemento conectado	Pin
MOSI	11
MISO	12
SCLK	13
SS	10

Arduino Pro Mini DC		
Elemento conectado	Pin entrada	Pin salida
Main Powe Supply	A3	D5
Calibrator Supply	A2	D4
Clock Synthesis Supply	A1	D3
Stage One Supply	A0	D2
Stage Two Supply	A6	D6
Servidor de red SBC	A7	D7
Arduino Pro Mini Digital		
Elemento conectado	Pin entrada	Pin salida
PLL RF	A0	D3
PLL IF1	A1	D4
PLL IF2	A2	D5
PLL IF3	A3	D6
PLL IF4	D3	D7
PLL Calibrator	A6	D8
Arty FGPA	A7	D9
Arduino Pro Mini Digital		
Elemento conectado	Pin entrada	Pin salida
PLL RF	A0	D3
Satélite 1	A1	D4
Satélite 2	A2	D5
Satélite 3	A3	D6
Satélite 4	A6	D7
RF Calibrator	A7	D2

Tabla 12.-Pines de entrada y salida del Sistema de Control. Fuente: Elaboración propia.

Diseño PCB

En vista del número de dispositivos y cableado existente en el sistema de control diseñado, es necesario diseñar un sistema de integración para un mejor funcionamiento del sistema. Con este objetivo, se ha diseñado una placa de circuito impreso (*PCB*). Las reglas de diseño que se siguen para la creación de la placa son las siguientes:

- Las pistas deben de ser lo más cortas y directas posibles.
- Agrupar los componentes relacionados.
- Evitar las pistas con ángulos de 90 º.
- Utilizar pistas con tamaño mínimo de 1 mm.

El proceso seguido para el diseño se detalla a continuación:

- Se diseña el esquemático del circuito, con los componentes necesarios y las conexiones pertinentes.
- Se procede a la colocación y conexión de los componentes sobre la placa PCB.
- Se reordenan los componentes siguiendo las reglas comentadas y se obtiene la placa PCB.

En la figura siguiente se presenta el circuito impreso diseñado.

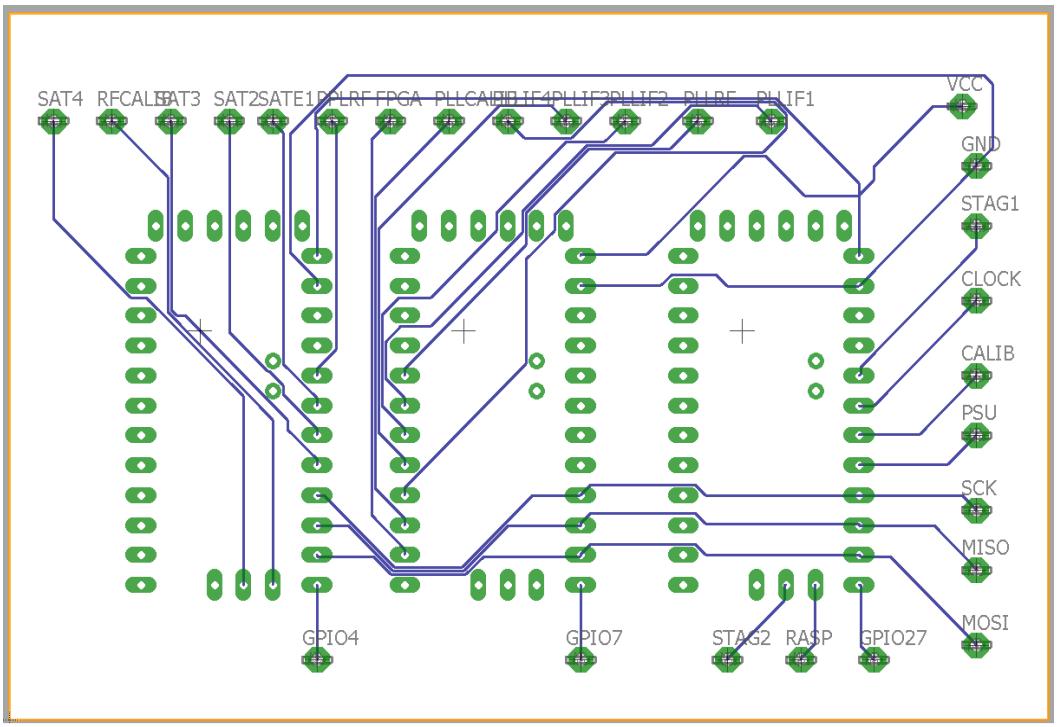

Ilustración 20 .-Diseño PCB del circuito de control. Fuente: Elaboración propia.

3.3. Diseño Software

En este apartado se describe el diseño del sistema software desarrollado. La estructura del sistema se ha dividido en tres partes diferenciadas:

- Software de Control: Se encarga de la adquisición, procesado y control.
- Arquitectura de comunicación: Se encarga de transmitir la información de desde el software de controla a la interfaz gráfica.
- Interfaz Gráfica: Se encarga de iniciar el sistema de control, la recepción y la representación de datos.

En el siguiente diagrama se muestra cada parte del sistema software, detallando las funciones que realiza cada uno de ellos.

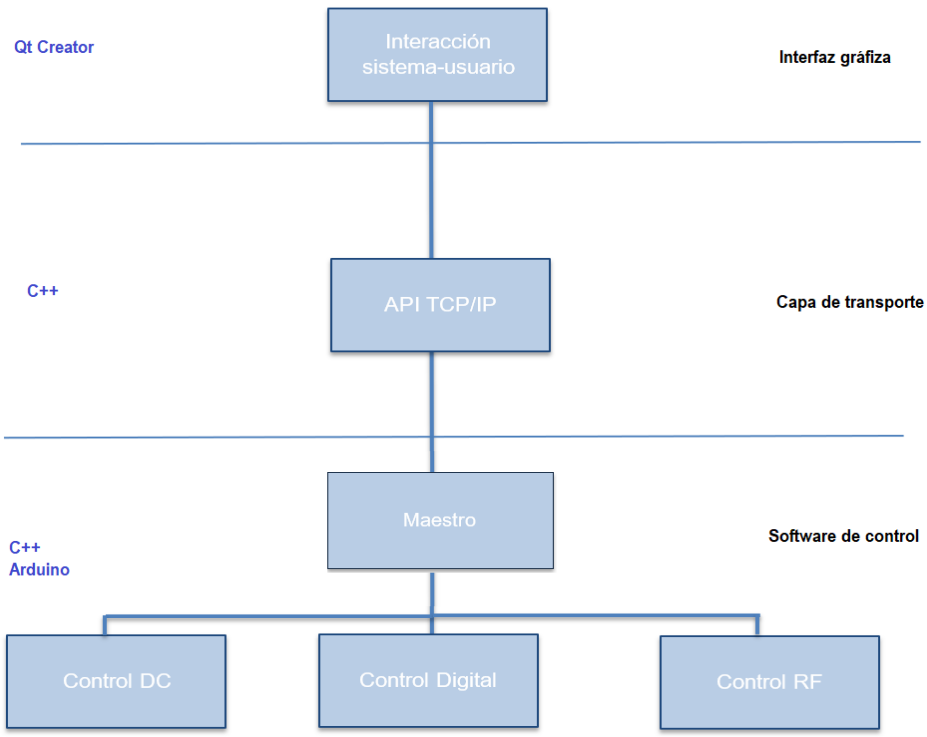

Ilustración 218.-Estructura Sistema de Software. Fuente: Elaboración propia.

En las siguientes secciones se desarrollará cada bloque descrita y cada módulo que las componen.

3.3.1. Arquitectura de Comunicación

En el diseño de un sistema distribuido, donde los dispositivos están separados físicamente entre sí, pero conectados mediante una red de comunicaciones, es necesario pensar en el modelo de red, es decir, qué máquinas formarán parte de la red y qué comunicaciones hay entre ellas. En este aspecto, existe una arquitectura de comunicación ampliamente utilizada que se basa en una en red centralizada con tres roles diferentes:

Ilustración 22.-Estructura Cliente-Maestro-Cliente. Fuente: Elaboración propia.

- **Cliente**: Máquina que dispone de los datos para su manipulación y representación. Se comunica con el maestro, pero nunca con los esclavos.
- **Maestro:** Máquina que hace el rol de servidor. Se encarga de recibir los trabajos del cliente y de distribuirlos entre los distintos esclavos. Actúa de intermediario, enviando los datos de los esclavos al cliente.
- **Esclavo:** Rol de las máquinas que disponen de las herramientas para la extracción de datos y control de dispositivos. Se comunican con el maestro, recibiendo de este los dispositivos a controlar y enviando de vuelta información sobre estos dispositivos.

El objetivo de esta arquitectura en tres niveles es la de permitir la cooperación entre múltiples clientes y múltiples esclavos, con la figura del maestro intercediendo entre ambos grupos y coordinando el uso de los esclavos por parte de los clientes mediante la

distribución de trabajos. La solución que se ha desarrollado en este trabajo se muestra en la siguiente figura.

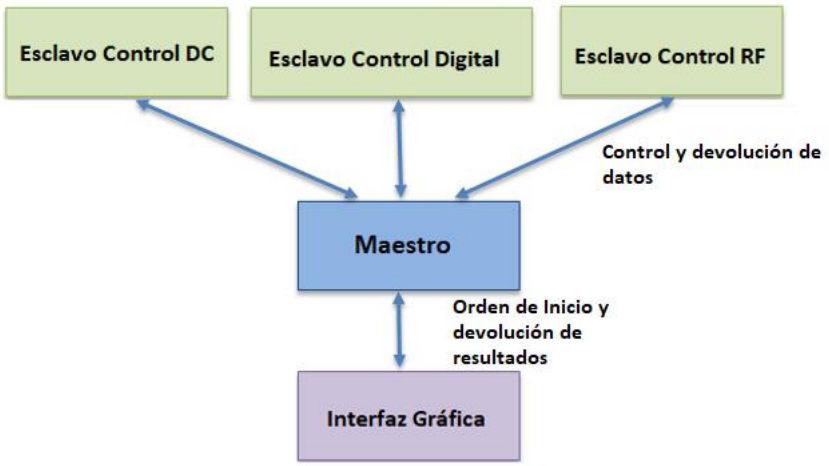

Ilustración 23.-Estructura Cliente-Servidor-Esclavo Implementada. Fuente: Elaboración propia.

Los dispositivos se comunican entre sí mediante la transmisión de mensajes, los cuales pueden ser de diferentes tipos. En nuestro proyecto, se requieren claramente dos tipos de comunicaciones diferentes; comunicación Maestro-Interfaz gráfica y comunicación Maestro- Esclavos. En los apartados siguientes, se explica el diseño y la implementación de cada uno de ellos.

3.3.1.1. Protocolo de Comunicación Maestro-Interfaz Gráfica

Para poder establecer un canal de comunicación entre la interfaz gráfica (cliente) y el maestro (servidor), es necesario hacer uso de algún protocolo de transporte. Los dos protocolos de transporte más significativos son:

- **Protocolo UDP** (*User Datagram Protocol*). Se trata de un protocolo de transporte no fiable puesto que no ofrece ninguna garantía sobre la integridad de los paquetes, ni sobre la llegada de los paquetes en el mismo orden que fueron enviados. De hecho, no garantiza siquiera su llegada. Tiene la ventaja de que, al no incluir las garantías anteriores, es más eficiente.
- **Protocolo TCP** (*Transmission Control Protocol*). Se trata de un protocolo de transporte fiable puesto que implementa mecanismos para garantizar la integridad

y llegada en orden de los mensajes. Al implementar estas garantías, el protocolo hace uso de comprobaciones adicionales y, por tanto, es menos eficiente que UDP.

El protocolo de transporte utilizado en la implementación de la comunicación en este proyecto es *TCP* por la razón ya comentada: fiabilidad en la transmisión de datos. No obstante, el protocolo de transporte TCP solo se encarga de la transmisión y recepción de datos entre dos procesos. Esto quiere decir que es necesario implementar un protocolo de aplicación. Para ello se han utilizado ***sockets.*** Un *socket* [15] es un objeto que debe ser configurado con los parámetros de la conexión (como mínimo: protocolo a usar, dirección de red del proceso remoto y número de puerto remoto) y que, a partir de entonces, se puede utilizar para establecer una comunicación con dicho proceso.

Diseño del protocolo de aplicación

Con el objetivo de que los módulos cliente y servidor sean capaces de comunicarse entre sí, es necesario establecer un protocolo de aplicación que defina la lógica, los mensajes intercambiados entre el Maestro(servidor) y la interfaz gráfica (cliente) y su orden. En la siguiente imagen se muestra de manera gráfica la secuencia de operación:

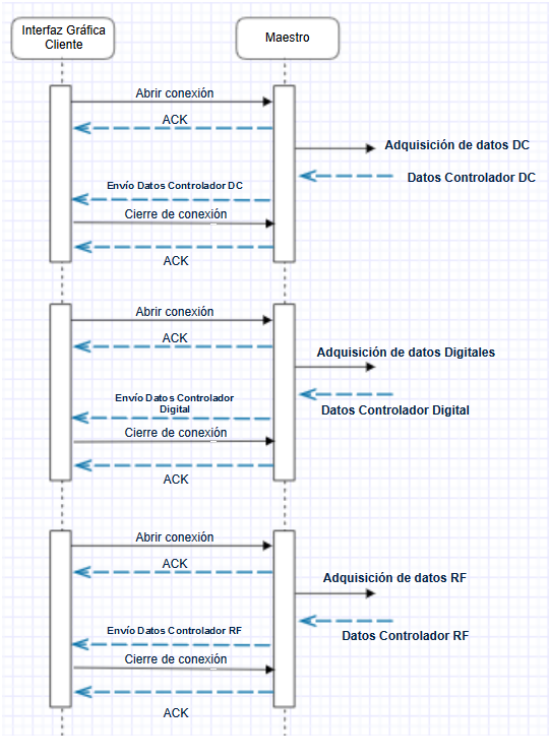

Ilustración 24.-Protocolo aplicación diseñado Maestro-Cliente. Fuente: Elaboración propia.

La secuencia de pasos ideada para la comunicación es la siguiente:

- El cliente decide inicializar el proceso de control abriendo una nueva conexión *TCP* (bidireccional) con el servidor. El servidor contesta a la petición indicando que está disponible para el control del sistema.
- El servidor realiza una petición al esclavo DC, indicándole la adquisición de datos. El esclavo responde con los datos adquiridos. Estos datos son enviados al cliente. El Cliente cierra la conexión. Este proceso dura 5 s.
- Se realiza el mismo proceso con los controladores Digitales y de RF. Estos procesos duran cada uno 5 s, por tanto, el tiempo total de adquisición de datos dura 15 s.
- El sistema queda en bucle, hasta que el usuario decide pausar el sistema de control.

Para hacer posible el protocolo diseñado, es necesario implementar las funcionalidades que se encargarán de la transmisión de los mensajes. Para ello se ha utilizado la siguiente metodología basada en el modelo cliente-servidor.

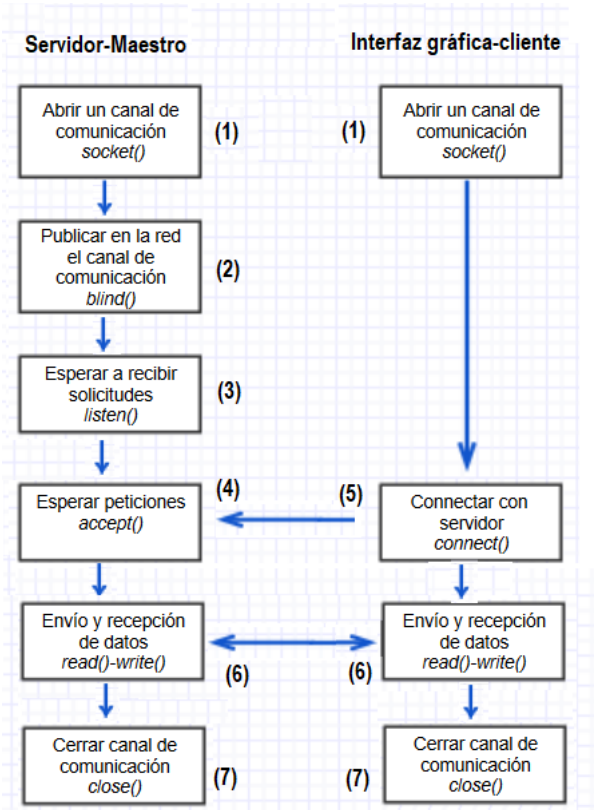

Ilustración 25.-Metodología sockets basada modelo cliente-servidor. Fuente: Elaboración propia.

La implementación de la estructura de la ilustración 25, es la siguiente:

1. El cliente y el servidor inician una conexión mediante la creación de un nuevo *socket.* El *socket* devuelve un descriptor para llamadas al sistema. La llamada es de la siguiente forma:

```
socket (int domain,int type, int protocol)
```

   - *Domain*: Familia de protocolos que se utilizará. En nuestro caso se establece como *AF_INET* para usar protocolos a través de internet.
   - *Type:* Clase de socket que se utilizará (Flujos o Datagramas). En nuestra aplicación se utilizará sockets de Flujo, por tanto, se debe especificar con *SOCK_STREAM.*
   - *Protocol:* En este campo se especifica qué tipo de protocolo se quiere utilizar. Como se ha comentado anteriormente, se utilizará protocolo TCP.

2. Aunque con el paso anterior se ha creado un *socket,* lo único que se han creado han sido las estructuras internas necesarias para mantener la comunicación. El siguiente paso es rellenar esta estructura interna, para ello se utiliza la función blind*().* Tiene el siguiente aspecto:

```
bind(int fd, struct sockaddr *my_addr, int addrlen)
```

   - *fd:* Es el descriptor del fichero *socket* devuelto por la llamada a *socket().*
   - *my_addr:* Es un puntero a la estructura *sockaddr.* Esta estructura contiene la información la dirección del protocolo de internet(IP).
   - *addrlen*: Contiene la longitud de la estructura *sockaddr* a la cuál apunta el puntero *my_addr.*

3. Después de ejecutar *blind()* es necesario llamar a *listen()* para que el *socket* sea marcado por el sistema como listo para recibir información. Esta llamada tiene una doble función, por un lado, pone al *socket* en modo "*standby*" a la espera de conexiones. Por otro lado, fija el tamaño máximo de la cola de peticiones para ese *socket.* La sintaxis es la siguiente:

```
listen(int fd,int backlog)
```

- *fd:* Fichero descriptor del *socket().*
- *Backlog:* número de conexiones permitidas. Al disponer de un solo cliente, la interfaz gráfica, este valor será 1.

4. Cuando el cliente intenta conectarse al servidor, se utiliza la función *accept().* El cliente sólo podrá conectarse al servidor si éste acepta. La sintaxis de esta función es la siguiente:

```
accept( int fd, void *addr, int *addrlen)
```

- *fd:* Fichero descriptor del devuelto por la función *listen().*
- *addr:* Puntero a la estructura *sockaddr* en la que se puede determinar a qué nodo se puede conectar el servidor y desde que puerto.
- *addrlen:* Longitud de la estructura de *addr.*

5. El cliente, una vez ejecutado *bind,* debe intentar establecer conexión con el servidor. Para ello debe de conocer la dirección IP del servidor y el puerto en el cual está escuchando. Esta tarea la realiza la función *connect().* Tiene el siguiente aspecto:

```
accept( int fd, void *addr, int *addrlen)
```

- *fd:* Fichero descriptor devuelto por la llamada a *socket().*
- *serv_addr.* Es un puntero a la estructura *sockaddr* la cual contiene la dirección IP destino y el puerto.
- *addrlen.* Longitud de la estructura de serv_*addr.*

6. Establecida la conexión entre el cliente y servidor las dos partes pueden enviar y recibir información. Con este propósito, se utilizan las siguientes funciones:

```
send( int fd, const void *msg, int len,int flags)
```

- *fd*: Es el fichero descriptor del socket, con el cual se desea enviar datos.

- *msg:* Es un puntero apuntando al dato que se quiere enviar.
- *len:* es la longitud del dato que se quiere enviar (en bytes).
- *flags:* Es un entero que indica opciones avanzadas, en nuestro caso se deja a 0.

```
recv(int fd, void *buf, int len, unsigned int flags)
```

- *fd*: Es el fichero descriptor del socket, con el cual se desea leer datos.
- *buf:* Es el búfer en el cual se guardará la información a recibir.
- *len:* longitud máxima de buffer (en bytes).
- *flags:* Número entero para indicar opciones avanzadas, en nuestro caso se deja a 0.

7. Finalmente, cuando el cliente y el servidor han establecido conexión e intercambiado información, se debe de cerrar el canal de comunicación. Esta tarea es llevada a cabo por la función *close(),* se elimina el fichero descriptor del *socket* creado:

```
close(int fd)
```

- *fd*: Es el fichero descriptor del socket creado.

En los siguientes apartados se explica la implementación de la estructura de comunicación.

Implementación protocolo lado del cliente

En esta sección se describe la relación de clases que se han diseñado para la implementación del protocolo de la parte de interfaz gráfica (cliente) programada en C++. En la siguiente figura se presenta el diagrama de clases de esta parte del sistema.

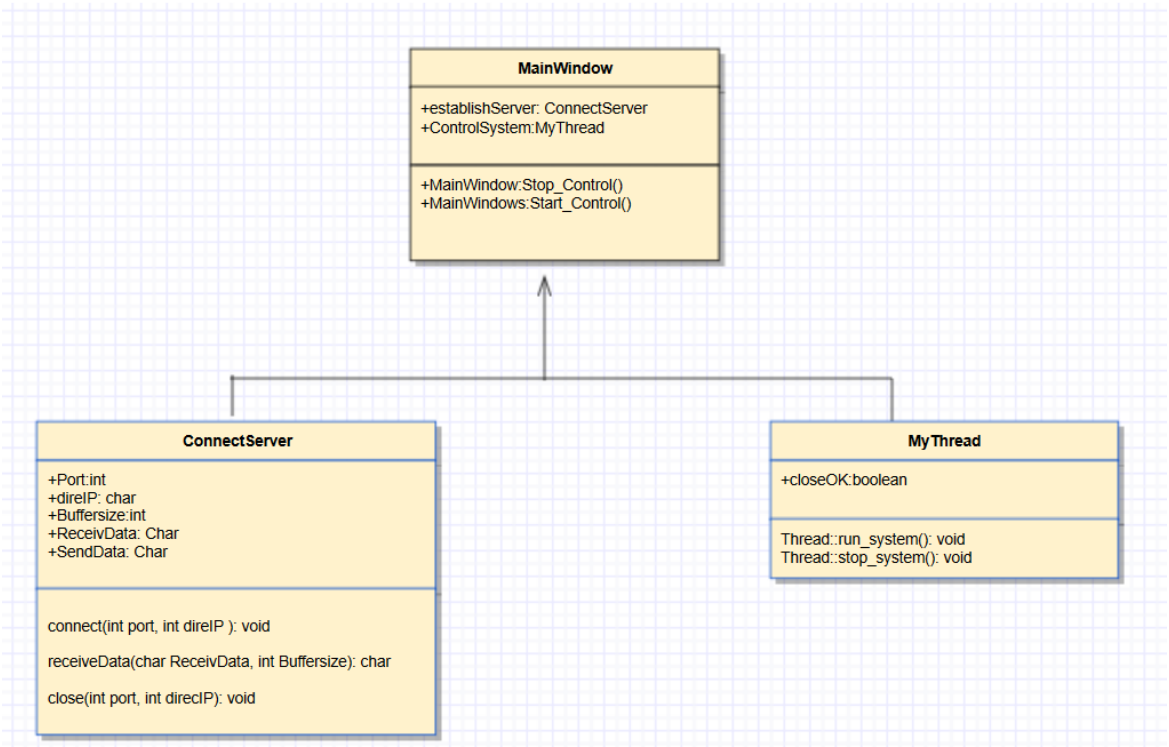

Ilustración 96.-Diagrama de clases de Cliente. Fuente: Elaboración propia.

MainWindows es la clase principal que controla el comportamiento de la interfaz gráfica. En el diagrama anterior no se han representado todas las clases, métodos y atributos que tiene esta clase, solo se han representado las clases del protocolo de comunicación. Esta clase esta tiene dos métodos que controlan el inicio y el fin del sistema de control:

- Método *Start_Control()*: es invocado cuando el botón "*Start*" de la interfaz gráfica es presionado. Este botón utiliza los métodos de clase *ConnectServer* para iniciar la comunicación y recibir información. También utiliza los métodos de la clase *MyThread* para iniciar el sistema de control
- Método *Stop_Control():* es invocado cuanto el botón "*Stop*" de la interfaz gráfica es presionado. Este botón utiliza el método *stop_system()* de la clase *Mythread* para indicar el fin del sistema de control.

ConectServer es la clase que establece la conexión entre la interfaz gráfica y el maestro. Esta clase tiene tres métodos:

- *Connect():* Se encarga de establecer conexión con el servidor.
- *receiveData():* Se encarga de recibir los datos del servidor.
- *Close():* Se encarga de cerrar la conexión creada.

MyThread es la clase que se encarga de iniciar y cerrar conexión con el servidor. El funcionamiento de esta clase se explica en el apartado 3.2.3 Interfaz gráfica. Está constituida por los siguientes métodos:

- *run_system():* se encarga de iniciar el sistema de control y por tanto el inicio de la conexión.
- *Stop_system():* se encarga de detener el sistema de control y por tanto de cerrar conexión con el cliente, cuando el usuario así lo decide.

Diseño protocolo lado Servidor.

Tras haber explicado la relación de clases de la parte cliente, se hará lo propio con la parte del servidor. Al igual que en el caso anterior, solo se han representado las clases y métodos del protocolo de comunicación. En la siguiente figura se presenta el diagrama de clases.

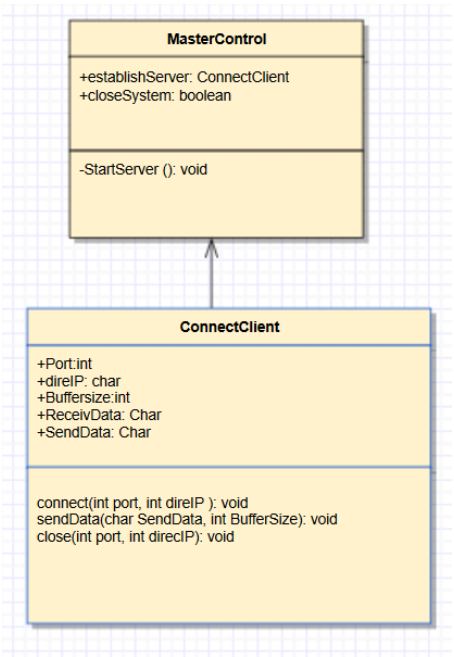

Ilustración 27.-Diagrama de clases servidor. Fuente: Elaboración propia.

MasterControl es la clase principal cuya función es comunicarse con el cliente y los esclavos. Esta clase tiene el siguiente método:

- *StartServer():* Este método se encarga de iniciar y controlar el sistema de control. Este método permanece en bucle hasta que el cliente decide terminar con el sistema de control

ConnectClient es la clase que se encarga de crear la conexión de comunicación con el cliente y comprobar si el usuario ha finalizado el sistema de control. Está constituida por los siguientes métodos:

- *Connect():* Establece conexión con el cliente.
- *sendData():* Envía los datos obtenidos por los esclavos.
- *Close():* Se encarga de cerrar la conexión con el cliente.

3.3.1.2. Protocolo de Comunicación Maestro-Esclavos

El protocolo *SPI* (*Serial Peripheral Interface*) es un estándar de comunicaciones, usado para transferir información entre circuitos en un equipo electrónico. Este protocolo tiene interfaz serie, que controla cualquier dispositivo electrónico digital que acepte un flujo de bus serie regulado por un reloj (síncrono).

El estándar *SPI* tiene las siguientes características:

- La comunicación se realiza siguiendo el modelo maestro/esclavo.
- El bus es full dúplex, es decir, se puede enviar y recibir información de manera simultánea, lo cual eleva la tasa de transferencia de datos.
- Velocidad de comunicación relativamente elevadas.
- No existe un medio de control de flujo de datos.

El bus SPI se define mediante cuatro pines:

- *SCKL*: Señal reloj del bus. Esta señal rige la velocidad a la que se transmiten los bits.
- *MOSI* (*Master Output Slave Input*): Salida de datos del maestro y entrada de datos del esclavo.

- *MISO* (*Master Input Slave Output*): Salida de datos de los esclavos y entrada de datos al maestro.
- *SS* (*Slave Select*): Señal que habilita el integrado al que se envían los datos.

El ciclo de funcionamiento es el siguiente:

1. Se habilita el chip al cual hay que enviar la información mediante un bus de habilitación.
2. Se carga el buffer de salida.
3. El reloj comienza a generar una señal cuadrada, en el que por cada flanco se pone un bit en MISO.
4. El receptor en cada flanco de subida captura el bit de la línea MISO y lo incorpora al buffer.
5. Acabado de transmitir toda la información, la línea de habilitación pasa a un estado de reposo.

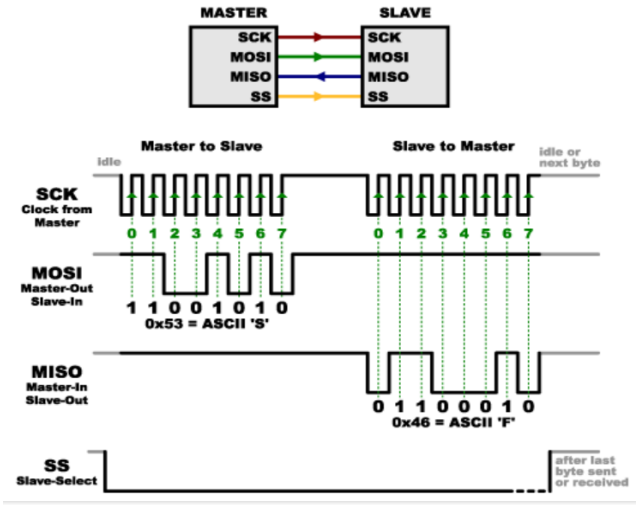

Ilustración 108.-Conexión y ciclo de funcionamiento de SPI. Fuente: https://learn.sparkfun.com

El proceso descrito se repite 8 veces para transmitir un byte. Se ha de tener en cuenta que el maestro y el esclavo envían y reciben datos simultáneamente.

En los siguientes apartados se explicará el diseño y la implementación de la comunicación entre el maestro y los esclavos.

Características bus SPI en Raspberry-Pi y Arduino

El estándar *SPI* no define un protocolo para la transmisión de los datos y pese a que especifica el número de bits que componen cada dato a transmitir, existen muchos dispositivos *SPI* compatibles que no emplean exactamente la definición. Ante esto, se hace indispensable recurrir a las especificaciones del dispositivo utilizado.

Arduino Pro Mini

Internamente el microcontrolador *ATmega328* tiene un bus *SPI* que puede trabajar como maestro o esclavo. Tiene las siguientes características:

- Operación Full-Dúplex.
- Modo Máster o Esclavo
- Operación con doble buffer con registro de transmisión y recepción por separado.
- Tamaño máximo de transmisión 8 bits.
- Programación del orden de recepción y transmisión.
- Frecuencia máxima en modo maestro (module *clock*/2)
- Frecuencia máxima en modo esclavo (module *clock*)
- Interrupción de transmisión y recepción.

Para manejar internamente este bus, se hace uso de una serie de registros:

- *SPCR* (Registro de Control *SPI*): Este registro es básicamente el registro maestro, es decir, contiene los bits para inicializar el SPI y controlarlo.
- *SPSR* (Registro de estado del *SPI*): Este es el registro de estado. Este registro se utiliza para leer el estado de las líneas de autobús.
- *SPDR* (Registro de Datos *SPI*): Este es el registro de lectura/escritura donde tiene lugar la transferencia real de datos.

Raspberry Pi mode B+

El bus *SPI* del microcontrolador *BCM2835* posee las mismas características que *ATmega328* exceptuando el tamaño del bus, configurable a 16 bits. Para manejar internamente este bus, se hace uso de los siguientes registros:

- *CS* (Registro de Control y estado): Este registro contiene los bits de control y estado del bus *SPI.*
- *FIFO* (Registro de Datos *SPI*): Este es el registro de lectura/escritura.
- *CLK* (Registro *Clock SPI*): Este es el registro que controla la sincronización del bus *SPI*.
- *DLEN (*Registro de tamaño de datos *SPI):* Este registro controla el tamaño de los datos enviado y recibido del bus *SPI*

Diseño

La metodología diseñada para el intercambio de información entre el maestro y los esclavos es la mostrada en el siguiente diagrama.

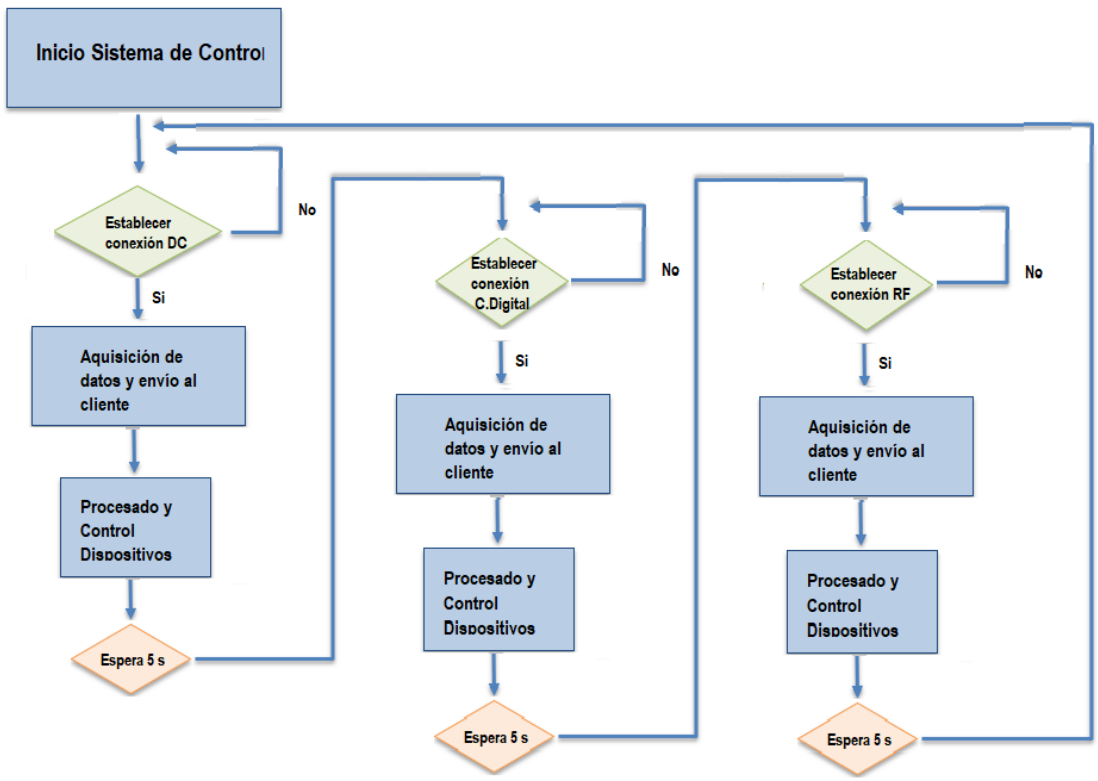

Ilustración 119.- Metodología diseñada para intercambio de información para el bus SPI. Fuente: Elaboración propia.

- El maestro inicia el sistema de control.
- El maestro establecer conexión con el esclavo que controla los dispositivos DC. Si la conexión no se realiza correctamente, cada 10 milisegundos intentan establecer una nueva conexión hasta obtener respuesta del controlador DC.
- Establecida la conexión, el controlador DC adquiere y envía datos de los dispositivos.
- El controlador DC procesa los datos y actúa en función del estado de los dispositivos.
- El maestro espera 5 s y establece conexión con el siguiente controlador. Se realiza el mismo proceso descrito en los pasos anteriores.
- Finalizado el último proceso de control (controlador RF) . El maestro regresa al estado original e inicial nuevamente el proceso descrito.

Descrito el funcionamiento general de comunicación entre maestro y esclavos, el siguiente paso consiste en establecer el protocolo de aplicación que defina la lógica, los mensajes intercambiados y su orden. En el siguiente diagrama se muestra el protocolo diseñado con la secuencia de operación.

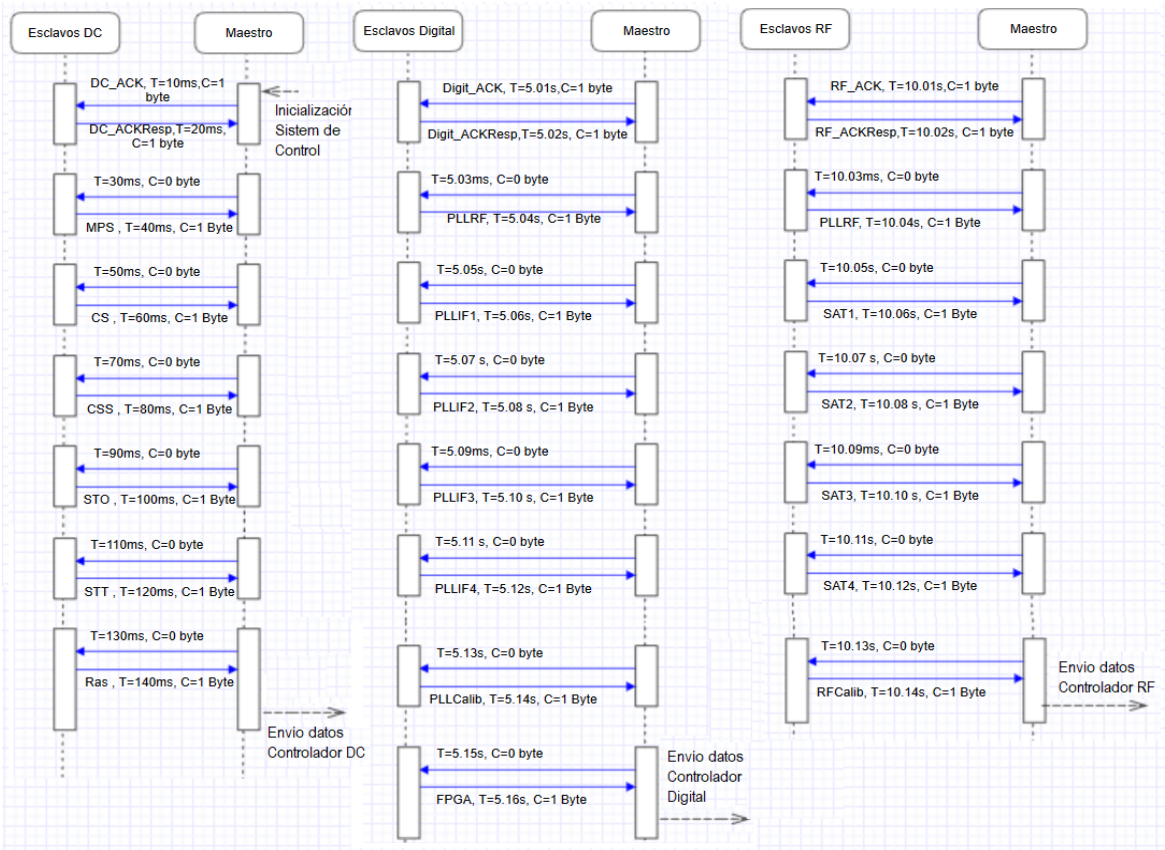

Ilustración 30.-Protocolo de aplicación SPI maestro-esclavos. Fuente: Elaboración propia.

- El Maestro inicia la comunicación con el esclavo DC enviando un byte de *ACK*. Éste responde con otro byte de confirmación. Este proceso queda en bucle hasta que la secuencia se ejecuta con éxito.
- Establecida la conexión, el intercambio de mensajes se hace de manera sincronizada. El maestro envía ceros, para obtener los datos de los dispositivos adquiridos por los esclavos. La secuencia de intercambio de mensajes se realiza cada 10 ms.
- Finalizado el intercambio de mensajes con el primer esclavo, el maestro espera 5 segundo y reanuda el mismo proceso con el siguiente esclavo.

En el siguiente apartado se describe la implementación para llevar a cabo el intercambio de mensajes.

Implementación

Con el objetivo de explicar la implementación para el intercambio de mensajes se ha divido esta parte en dos secciones, implementación por parte del maestro e implementación por parte de los esclavos.

Esclavo

Se ha resumido la implementación en los siguientes pasos:

1. Consideraciones iniciales
    a. La interacción con el bus *SPI* se hace a través de los registros *SPCR*, *SPSR* y *SPDR.*
    b. No es necesario establecer la velocidad de reloj, puesto que la maneja el maestro.
    c. Existe una librería *SPI* de *Arduino,* pero desafortunadamente no tiene soporte para el modo esclavo. La recepción y envío de mensajes se realizan accediendo a los registros.
    d. La lectura y escritura de datos SPI se realiza a través de *SPDR*. A nivel hardware *SPDR* contiene un registro de desplazamiento de 8 bits y un búfer de recepción de 8 bits. Cuando el esclavo está recibiendo datos, esos datos se transfieren al registro de desplazamiento bit por bit, mientras que los 8

bits originales del registro se devuelven al maestro. Cuando se ha desplazado un byte completo al registro, éste se copia en el búfer de recepción. El búfer de recepción no se actualizará de nuevo hasta que se reciba el siguiente byte completo.

e. La coordinación entre los dispositivos se logra añadiendo un tiempo de esper de 10 ms por parte del maestro, esto es necesario, puesto que *Arduino* necesita una pequeña ventana de tiempo para poder cargar el registro *SPDR* con el siguiente byte a enviar al maestro.

2. Configuración inicial:
    a. El pin MISO se ha de configurar como salida.
    b. Se ha de habilitar el bit de inicio *SPCR*.
3. Modo "espera": Como el maestro inicia la comunicación desplazando un byte al esclavo, éste requiere de un método para reconocer dicho byte. Existen dos métodos para el reconocimiento: *Interrupts* o *Polling.* En este proyecto se ha decidido utilizar el método *Polling.* Este método consiste en comprobar periódicamente el registro *SPCR,* que indica la recepción de un byte. Se ha implementado la función *ListenMode()* utilizando el método comentado.
4. Modo transferencia de datos.
    a. ACK: El maestro envía un *byte* "c" al esclavo, éste reconoce el *byte* y responde con un *byte* 'a'. Ambos dispositivos se colocan en bucle hasta que la secuencia haya sido ejecutada correctamente. Esta técnica aborda de manera efectiva dos problemas:
        - En el bus SPI, el maestro no tiene forma de confirmar que el esclavo está realmente en línea y recibiendo información. Si el esclavo estuviera fuera de línea, el maestro enviaría datos y obtendría respuesta como una cadena de ceros o cualquier ruido aleatorio presente en la línea *MISO*.
        - La otra cuestión, es la posibilidad de que el esclavo pierda un byte en algún punto del camino o empiece a transmitir en medio de una comunicación.

b. En el momento en que el esclavo y el maestro quedan enlazados, se inicia la transmisión de datos según el protocolo de la ilustración 30.
c. Para sincronizar el intercambio de mensajes se ha utilizado la estructura de control *switch-case* con una variable *marker*. La variable *marker* se utiliza para mantener un registro de los mensajes a transmitir.
d. El proceso descrito se implementa en el la funciones *sendDC(), sendRF()* y *sendDigital().*

Maestro

De la misma forma que en la sección anterior, se ha resumido la implementación en los siguientes pasos:

1. Consideraciones iniciales.
    a. El sistema operativo que se utiliza en este proyecto, *Raspbian,* proporciona varias capas de abstracción del bus *SPI.* Para obtener acceso al bus, se abre un archivo virtual que apunta al dispositivo *SPI.* Las interacciones con el bus *SPI* se realizan leyendo y escribiendo en este archivo.
    b. La comunicación con el hardware *SPI* se realiza a través de la llamada al sistema *ioctl*(control de entrada/salida).
    c. El controlador *spidev* proporciona las funcionalidades para la configuración del sistema: polaridad del reloj, fase del reloj, ancho de bits de transmisión, orden y velocidad de transmisión.
    d. Para la transmisión de mensajes, el controlador *spidev* establece una estructura de datos *spi_ioc_transfer*, mediante la cual se puede recibir y enviar mensajes.
2. Configuraciones iniciales:
    a. En este proyecto se han dejado las configuraciones por defecto, exceptuando la velocidad de transmisión. Las especificaciones del microcontrolador de los esclavos *ATmega* indica que el reloj del sistema de esclavos debe ser cuatro veces menor que la velocidad de transmisión *SPI*. Puesto que estamos usando un reloj de 8 MHz la comunicación SPI puede ser de 2 MHz. En nuestra aplicación se ha establecido a 1 MHz.

b. Las configuraciones por defecto son las siguientes:
    - Tamaño transmisión de 8 bits.
    - El bit más significativo se transmite primero
    - Modo SPI 0. Este modo define en qué polaridad trabaja el bus SPI

3. Modo transferencia de datos: A diferencia de los esclavos, la implementación de la transmisión y recepción de datos es más sencilla en la parte del maestro, dado que no es necesario diseñar una estructura de control tipo *switch-case. S*encillamente, el máster transmite ceros con el objetivo de que el esclavo retorne los valores adquiridos de los dispositivos según el protocolo de la ilustración 30. Para la transmisión y recepción se ha implementado las funciones *spiTxRXDC(), spiTxRXDigital()* y *spiTxRXRF()* que utiliza la estructura *spi_ioc_transfer* mencionada anteriormente.

3.3.2. Software de Control

En esta sección se desarrolla la metodología diseñada para el control de los dispositivos y se explica, sin entrar en demasiado detalle, la implementación.

A la hora de desarrollar un sistema, es difícil mantener una idea del funcionamiento completo de este, pudiendo dar lugar a confusiones que desembocan en errores y problemas. Los diagramas de estados son útiles porque ayudan a describir el comportamiento del sistema de manera sencilla y de fácil entendimiento. El sistema de control se puede describir básicamente por dos procesos: proceso que realizan los esclavos y proceso que efectúa el máster, por lo que se ha creado un diagrama de estados para cada uno.

3.3.2.1. Diseño parte Esclavos

El diagrama de estado que gobierna el proceso llevado a cabo por los esclavos es el siguiente:

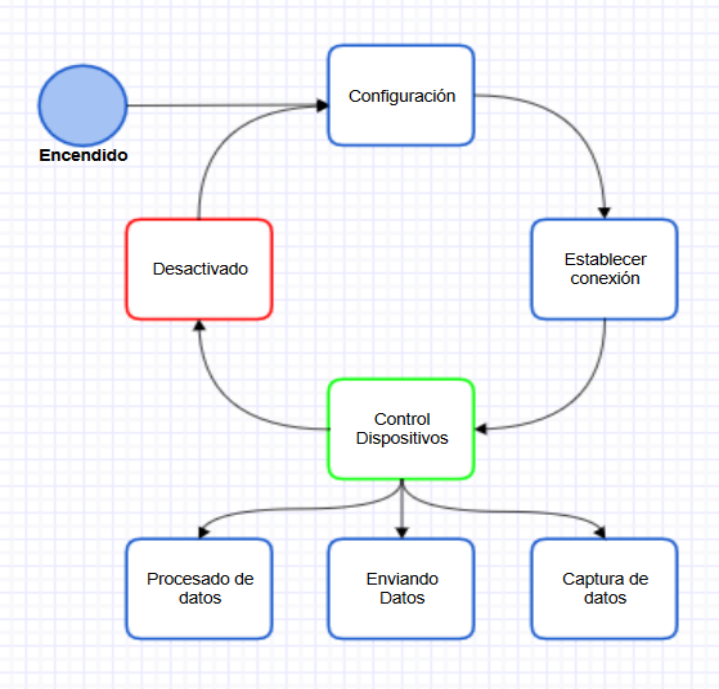

Ilustración 31.-Diagrama de estados funcionamiento de esclavos. Fuente: Elaboración propia.

1. Encendido: El dispositivo se enciende, inicializando la *IDE Arduino* y pasa al estado Configuración, comenzando la ejecución del programa.

2. Configuración: Se inicia el *sketch* de *Arduino*, realizando configuraciones básicas de entradas y salidas, declaración de variables, etc.
3. Establecer conexión: El dispositivo se encuentra a la espera de establecer conexión con el maestro, este proceso se realiza de manera permanente.
4. Control de dispositivos: El dispositivo ha establecido conexión y comienza a adquirir datos, enviarlos y procesarlos.
5. Desactivado: El dispositivo permanece en modo de espera hasta una nueva llamada por parte del maestro.

La estructura de programación de *Arduino* consta de dos funciones imprescindibles para el funcionamiento de éste. Se trata de la función *setup* y la función *loop*. La primera de ellas inicializa el programa, se ejecuta una sola vez al comenzar el programa. La segunda será donde se encuentre el código a ejecutar y se repetirá de manera cíclica.

En el diagrama de flujo de la ilustración 32, se representa con mayor detalle la secuencia de pasos que realiza el controlador DC. Los controladores Digitales y RF realizan el mismo proceso, con algunas variaciones ínfimas, de modo que la explicación que se detalla a continuación es aplicable a los dos controladores mencionados.

- El proceso entra en la función *Setup* donde se inicia el sistema configurando la comunicación *SPI* y la inicialización de los dispositivos a controlar.
- Configurado el estado inicial del dispositivo, el proceso salta a la función *loop* donde se ejecutan las funciones *ListenMode(), acquisitionDC(), sendDC()* y *resetDevice().* En el apartado anterior, se ha explicado el funcionamiento de *ListenMode*() y *sendDC(),* de modo que en esta sección se explicará las dos funciones restantes.
- *acquisitionDC()* se encarga de la lectura analógica de los dispositivos a controlar, esta función retorna un vector con los valores de los dispositivos controlados. La captura de datos está codificada como un número entero de 0 a 1023, para convertir estos datos a un valor de tensión se utiliza la función auxiliar *convertVolts().*

- *resetDevice()* se encarga de restablecer el funcionamiento de los dispositivos, para ello compara cada elemento del vector creado por la función *acquisitionDC()* con el valor de tolerancia indicado en las especificaciones.

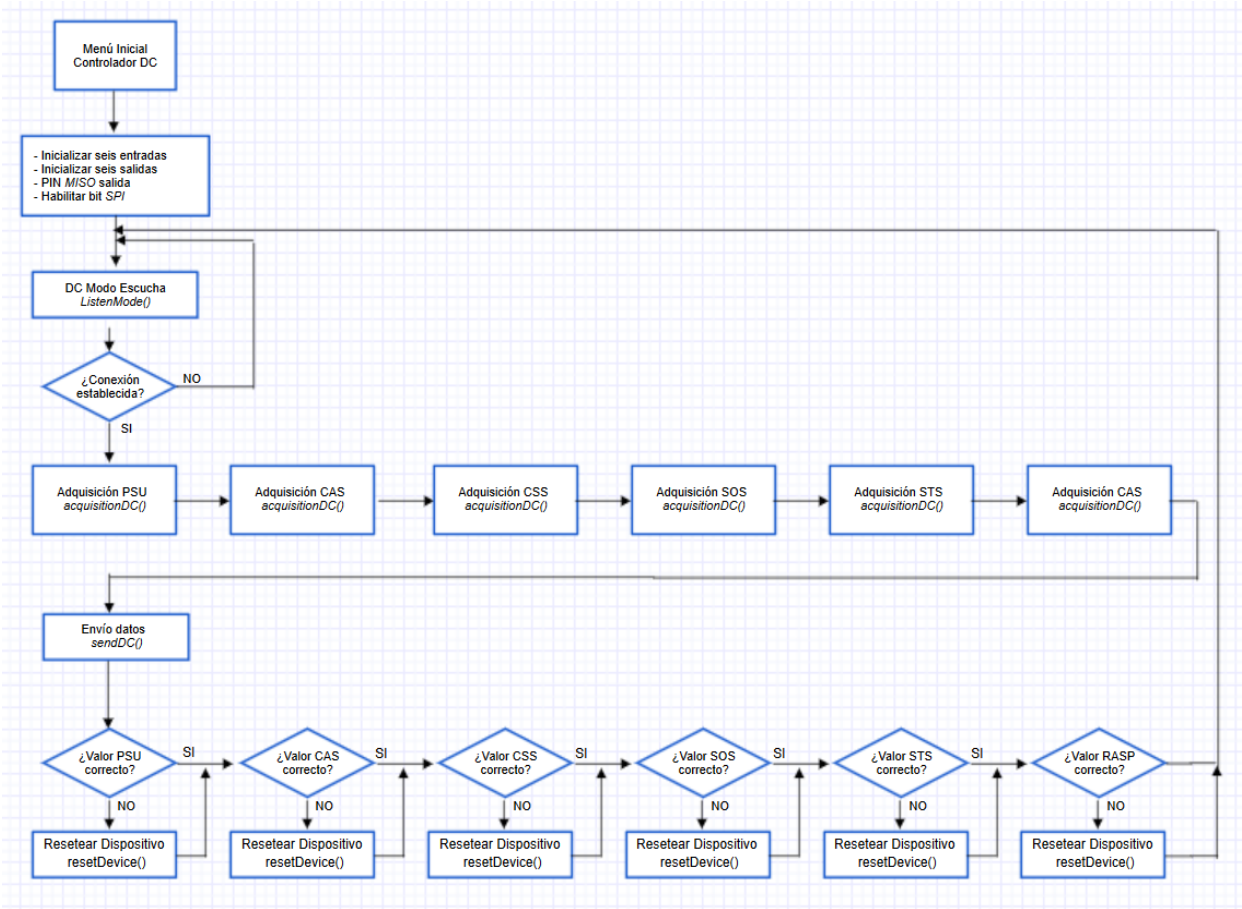

Ilustración 32.-Diagrama controlador DC. Fuente: Elaboración propia.

3.3.2.2. Diseño parte Maestro

El diagrama de estado que gobierna el proceso llevado a cabo por el maestro es el siguiente:

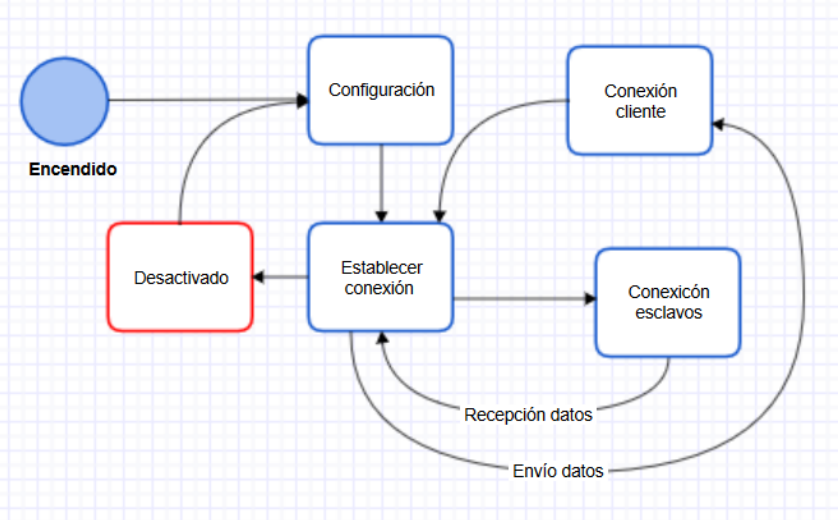

Ilustración 33.- Diagrama de estados funcionamiento de maestro.

1. Encendido: El maestro se enciende, inicializando *Linux Raspbian* y el *Script* de control.
2. Configuración: Se inicializan las librerías, las clases y los métodos empleados.
3. Establecer conexión: El cliente establece conexión con el maestro y éste, a su vez establece conexión con los eslavos. Los esclavos envían información al maestro y éste los transfiere al cliente.
4. Desactiva: El maestro permanece en modo espera cuando el cliente decide desactivar el sistema de control.

3.3.3. Interfaz gráfica

La importancia de desarrollar una interfaz gráfica de control reside en facilitar la detección y corrección de errores a medida que el sistema hardware de *GeoSAR* aumenta en complejidad. Con el desarrollo de esta interfaz, se pretende facilitar la tarea a los investigadores de *RSLAB*.

3.3.3.1. Requisitos de la interfaz Gráfica.

Aunque se ha definido inicialmente una serie de requisitos que debe de cumplir el sistema software, no se han especificado las funcionalidades y lo que el usuario debe de poder hacer una vez implementado. Los casos de uso son útiles para definir de manera visual las posibles funcionalidades y acciones que el usuario puede realizar dentro del sistema. Estos casos de uso están definidos en la siguiente figura.

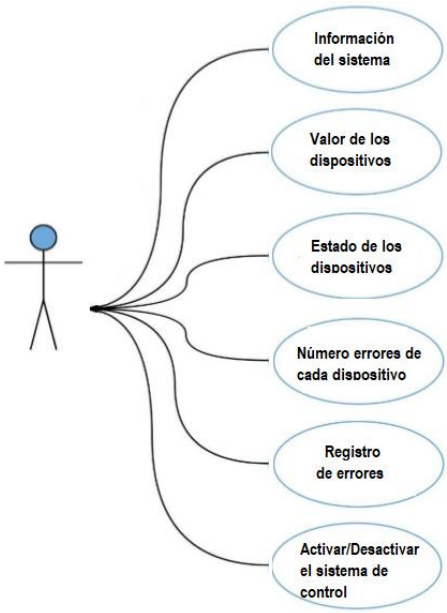

Ilustración 34.-Casos de uso Interfaz Gráfica. Fuente: Elaboración propia.

En las siguientes tablas se muestran requisitos del sistema:

Ver información de sistema
La página de inicio debe mostrar la siguiente información básica del sistema: -Funcionamiento general de la interfaz - Guía de usuario
Prioridad de requisito: Alta

Ver valor de los dispositivos
El sistema debe de permitir comprobar -Ver valor dispositivos DC real y nominal -Ver valor de dispositivos Digitales real y nominal -Ver valor de dispositivos RF real y nominal
Prioridad de requisito: Alta

Ver estado de los dispositivos
-El sistema debe de permitir comprobar: -Estado dispositivos DC -Estado dispositivos Digitales -Estado dispositivos RF
Prioridad de requisito: Alta

Ver errores de los dispositivos
El sistema debe de mostrar el número de errores durante las últimas 24 h de: -Dispositivos DC -Dispositivos Digitales -Dispositivos RF
Prioridad de requisito: Alta

Registro histórico de errores
El sistema debe de guardar en un fichero .exe los errores del sistema de: -Dispositivos DC -Dispositivos Digitales -Dispositivos RF
Prioridad de requisito: baja

Activar/Desactivar Sistema de control
El sistema debe de permitir al usuario activar y desactivar el sistema de control mediante la interfaz gráfica.
Prioridad de requisito: alta

Tabla 13.-Especificaciones de la Interfaz Gráfica. Fuente: Elaboración propia.

Especificados los requisitos, en los siguientes apartados se explica el diseño de la interfaz gráfica.

3.3.3.2. Diseño funcional

El funcionamiento de la interfaz gráfica se representa en el siguiente diagrama de estados.

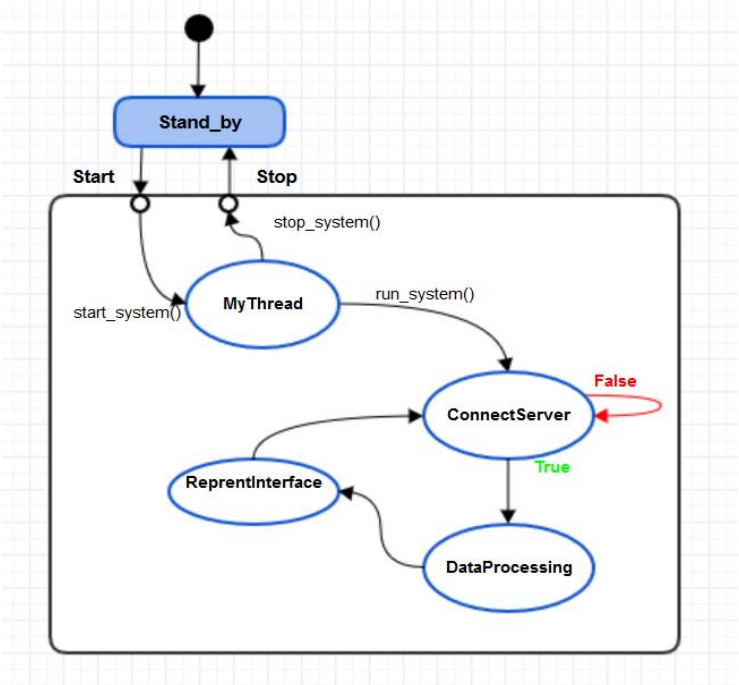

Ilustración 35.-Diagrama de estados Interfaz Gráfica. Fuente: Elaboración propia.

1. *Stand_by*: El primer estado se alcanza cuando el sistema de control no está en funcionamiento. En este estado el usuario puede iniciar el sistema pasando al siguiente estado.
2. *MyThread*: Este no es un estado en sí, pero es el elemento que controla los estados de la interfaz gráfica. Cuando el usuario inicia el sistema, se invoca a esta clase cuya función es crear un hilo de ejecución para iniciar el sistema de control y otro hilo para detener el sistema.
3. *ConnectServer*: En este estado la interfaz gráfica establece conexión con el maestro para iniciar el sistema de control y obtener datos de los dispositivos controlados. En proceso queda en bucle hasta establecer conexión con el maestro.
4. *DataProcessing*: En este estado se determina los errores y el estado de los dispositivos, asimismo, se guardan los errores en un fichero *.xlsx*.
5. *RepresentInterface*: En este estado, se muestran los datos recibidos, el estado y los errores de los dispositivos.

Cuando el proceso concluya, el sistema regresa al estado *ConnectServer* para iniciar nuevamente la recepción de datos. En todo momento del proceso, el usuario puede volver al estado *Stand_by* a través de la clase *MyThread.*

3.3.3.3. Lógica de Interfaz y conexión de señales

La estructura diseñada para la implementación de la interfaz sigue la organización del diagrama de estados de la ilustración 36. Cada estado se encuentra representado por una clase generada con la clase *QMainWindow.* Existe una cierta jerarquía donde el estado *Stand_by* representa la clase *MainWindow.* En la siguiente figura se muestra un diagrama en que se puede observar la jerarquía comentada.

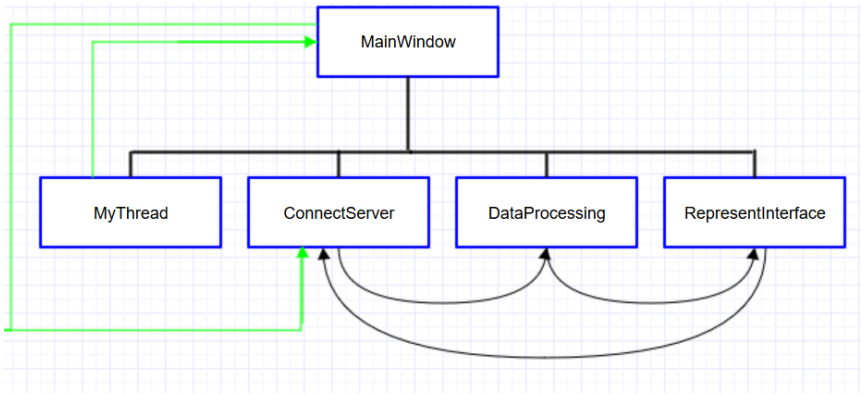

Ilustración 126.-Estructura Interfaz Gráfica. Fuente: Elaboración propia.

En la figura, las flechas de color negro señalan el orden de ejecución y las clases que se encuentras conectadas. No obstante, debido a la jerarquía que se sigue, las órdenes que se manifiestan entre las clases son asignadas y dirigidas por la clase *MyThread.* Estas órdenes tienen que pasar por la clase *MainWindow.* Para aclarar este punto, a modo de ejemplo, se han trazado las líneas de color verde que indican la lógica que se sigue: para acceder a la clase *ConnectServer,* se ha de emitir una señal desde la clase *MyThread,* esta señal conecta con *MainWindow,* que es la encargada de emitir una señal a *ConnectServer* indicando el inicio de la comunicación. Este método se aplica a cualquier comunicación entre clases.

Es importante aclarar la función que realiza la clase *MyThread* y porqué se ha utilizado en la implementación de la interfaz. Uno de los requisitos del sistema de control es la capacidad de funcionar durante largos periodos de tiempo, para lograr este requisito, la

interfaz gráfica y el software de control entran en un bucle en el que únicamente pueden salir si así lo decide el usuario. En el momento en que la interfaz entra en este bucle, la *API* de *QTCreator* crea únicamente un hilo principal de ejecución que procesa los eventos programados en este bucle. Por tanto, si se quiere realizar otra acción que no se encuentre dentro del bucle, es necesario crear manualmente otro hilo de ejecución que interactúe con el hilo principal. Con este objetivo, se ha creado la clase *MyThread,* cuya función consiste en interactuar con el hilo principal de ejecución para detener o iniciar el sistema de control.

3.3.3.4. Diseño visual e implementación

Descrita las relaciones entre las clases y la lógica de conexiones, a continuación, se detalla la implementación, sin entrar en detalle, y se relaciona con el aspecto visual de la interfaz. En el siguiente diagrama se observa las funciones empleadas y la relación entre ellas:

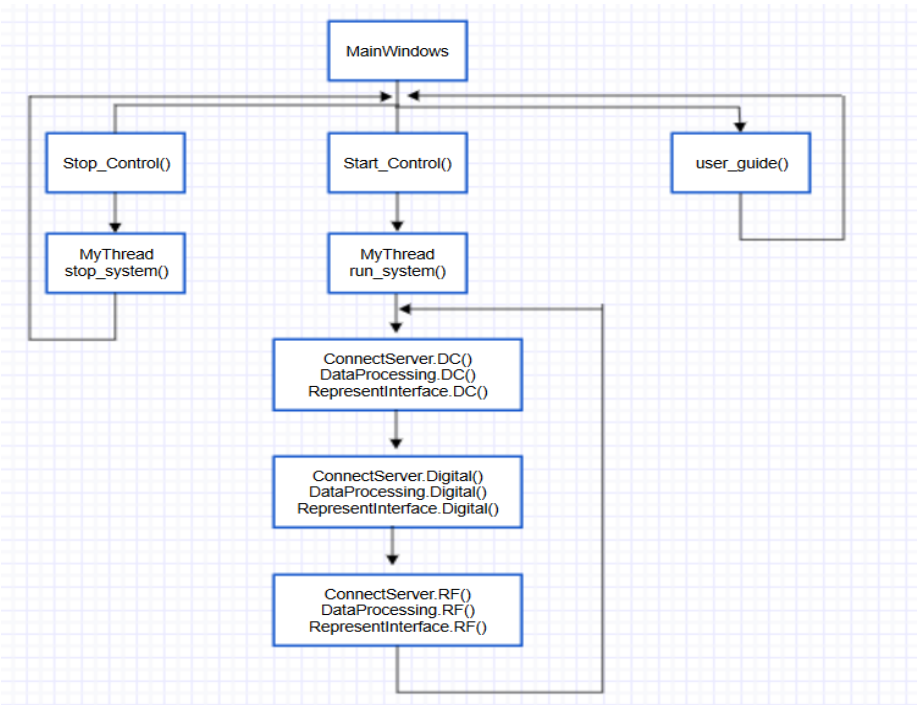

Ilustración 137.-Diagrama de funcionamiento de la Interfaz Gráfica. Fuente: Elaboración propia.

1. Se inicia la interfaz gráfica, configurando parámetros básicos de inicialización de librerías, clases, métodos y variables. En este punto del proceso el usuario puede realizar tres acciones, detener el sistema, inicializar el sistema de control u ojear las instrucciones de uso.

2. *Stop_Control()*: Este método se encuentra en la clase principal *MainWindow.* Está constituida por un botón de la clase *Qwidget PushButton.* Su función es detener el sistema de control cuando el usuario así lo decide. Cuando este botón es pulsado, se invoca al método *stop_system()* de la clase *MyThread,* que interactúa con el hilo principal de ejecución, indicando el fin del proceso.
3. *Start_Control()*: Al igual en el método anterior, se encuentra en la clase principal *MainWindow* y está constituido por un botón *PushButton.* Su función es inicializar el sistema de control, para ello, utiliza el método *run_system()* de la clase *MyThread.* Este método invoca a las siguientes clases:
    a. *ConnectServer*: Esta clase tiene tres métodos, *DC(), Digital(), RF().* Cada método se encarga de establecer conexión con el Maestro y recibir datos de los dispositivos DC, Digitales y RF respectivamente.
    b. *DataProcessing:* Al igual que la clase anterior, esta función tiene tres métodos, que se encargan de procesar los datos de los dispositivos DC, Digitales y RF.
    c. *RepresentInterface:* Esta clase se encarga de representa de forma visual los datos obtenidos, para ello tiene tres métodos *DC(), Digital() y RF().* La representación de datos se realiza mediante el mostrador de la clase *QWidget LineEdit.* Donde a cada mostrador se le asigna un *objectName* diferente. Los estados de los dispositivos se representan mediante la clase *Qwidget PushButton.* Se ha adaptado este botón para realizar la función de indicador de estado.
4. *User_guide():* Este método se encuentra en la clase principal *MainWindow.* Su función es proporcionar la información necesaria para añadir un nuevo controlador o un nuevo dispositivo al sistema de control. Está constituido a partir de botón *PushButton.*
5. La secuencia de pasos desde que el usuario pulsa el botón *Start* es la siguiente:
    a. Se crea el hilo de trabajo principal que ejecuta los eventos del método *Start_Control()*.
    b. Se establece conexión con el Maestro, se obtienen los datos de los dispositivos DC y se cierra la conexión.

c. Se procesan estos datos, determinado los dispositivos DC que operan incorrectamente y se guardan estos errores en un archivo *.xlsx.*

d. Se representan los datos, donde se indica el valor DC de cada dispositivo, su estado, el valor nominal y el número de errores durante las últimas 24 horas.

e. Se realiza el mismo proceso de captura, procesado y representación para los dispositivos Digitales y RF.

f. El usuario puede detener la ejecución del hilo principal en cualquier momento.

En la ilustración 38 se observa el aspecto visual de la interfaz gráfica diseñada. El primer módulo lo forman los botones *stop y start.* Estos botones al ser pulsados invocan a las funciones descritas *Start_Control()* y *Stop_Control()*. En el segundo módulo se representan los valores, el estado, el valor nominal y el número de errores de los dispositivos. Este módulo invoca a la clase comentada *RepresentInterface.* En el tercer módulo, *Settings* se encuentra la guía de uso y una breve explicación del objetivo de la interfaz.

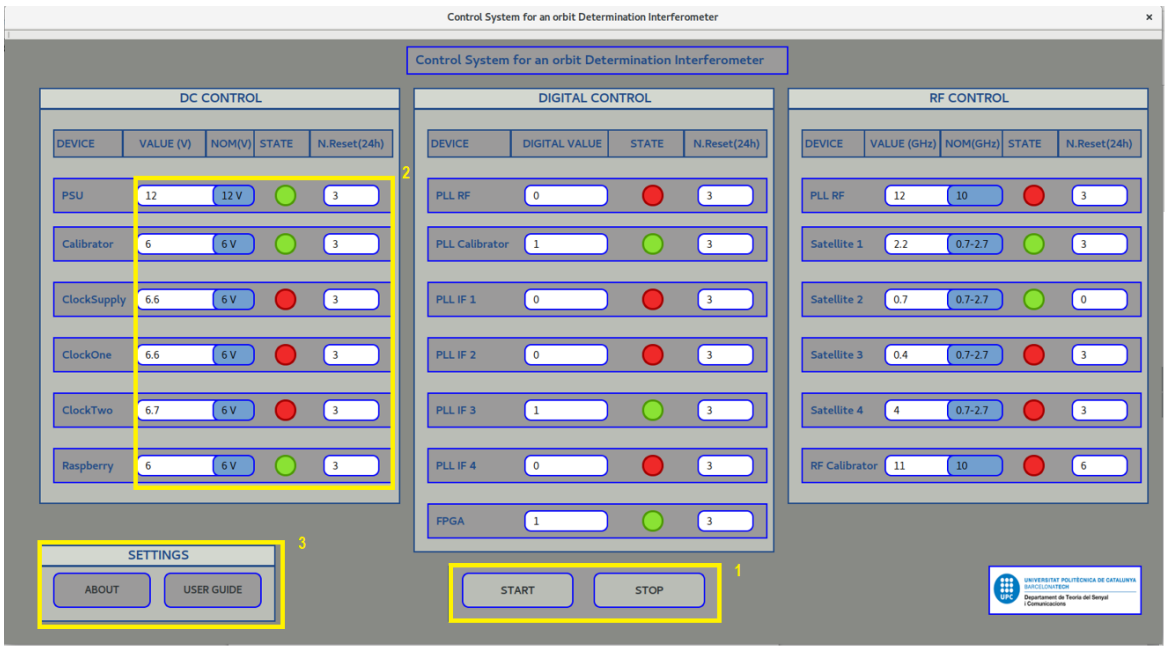

Ilustración 148.-Aspecto Interfaz Gráfica. Fuente: Elaboración propia.

4. Resultados

En este capítulo se comentan las pruebas más relevantes que se han realizado durante el desarrollo del proyecto para comprobar su correcto funcionamiento. Se explican las pruebas realizadas a la interfaz gráfica, al sistema hardware de control y al sistema de control completo.

4.1. Interfaz gráfica.

A continuación, se detallan las pruebas realizadas para comprobar el funcionamiento de la interfaz gráfica.

4.1.1. Prueba 1. Activar y detener la interfaz gráfica.

Descripción general

Al encender la interfaz gráfica, el usuario debe de poder realizar dos acciones, arrancar el sistema de control u ojear la guía de usuario. Si el sistema de control está en funcionamiento, el usuario debe de poder detener el sistema en cualquier momento. Con esta prueba, se comprueba la correcta programación de los botones *start, stop y userguide.* Además, se comprueba la correcta programación del hilo (*thread)* principal y secundario.

Procedimiento para la comprobación.

1. Encender la interfaz gráfica.
2. Utilizar el botón *userguid* y comprobar la ventana emergente con la guía de usuario.
3. Utilizar el botón *start* y comprobar el arranque del sistema de control.
4. Detener el sistema con el botón *stop* .
5. Iterar el paso 3 y 4.

Resultados

Se comprueba el correcto encendido de la interfaz, el funcionamiento de los botones *start*, *stop* y *userguide*.

4.1.2. Prueba 2. Representación correcta de los valores, el estado y los errores de los dispositivos controlados. Funcionamiento de la comunicación servidor-cliente.

Descripción general

El usuario debe de poder comprobar el valor de los dispositivos controlados, además del estado en el que se encuentran y el número de errores durante las últimas 24 h. Con esta prueba se comprueba la correcta representación de los datos, el procesado de datos para determinar qué dispositivos tienen un comportamiento incorrecto y la programación de la comunicación entre servidor-cliente.

Procedimiento para la comprobación.

1. Predefinir dos tantas de valores correctos e incorrectos, mezclados entre sí, para simular valores de los dispositivos DC, Digital y RF.
2. Inicializar el servidor en un ordenador con los valores anteriores.
3. Inicializar la interfaz gráfica.
4. Simular el proceso 60 veces.

Resultados

Al realizar esta prueba surge un inconveniente, aunque la interfaz gráfica establece conexión con el servidor correctamente y la transferencia de datos se realiza adecuadamente, la representación de datos no se ejecuta. Esto es debido al considerable número de eventos que debe de procesar la interfaz gráfica. Para solucionar este problema, se establece un tiempo de espera de 5s para cada nueva transferencia y representación de valores.

Resuelto el problema, se comprueba la correcta transmisión de datos por parte del servidor y la correcta recepción por parte de la interfaz. Los valores simulados se representan correctamente, de igual forma, que el estado y el número de errores.

4.1.3. Prueba 3. Funcionamiento durante amplios periodos de tiempo.

Descripción general

La interfaz gráfica debe de ser capaz de funcionar durante un periodo de tiempo amplio sin que se produzca ningún error de comunicación, representación o bloqueo del sistema. Además, con esta prueba, se verifica la correcta programación del temporizador de 24h y la función creada para exportar los errores de los dispositivos a Excel.

Procedimiento para la comprobación.

1. Predefinir dos tantas de valores correctos e incorrectos, mezclados entre sí, para simular valores de los dispositivos DC, Digital y RF.
2. Inicializar el servidor en un ordenador con los valores predefinidos.
3. Inicializar la interfaz gráfica.
4. Simular el proceso durante 32h.

Resultados

Tras simular el proceso durante 32h, se ha comprobado el correcto funcionamiento de la interfaz gráfica y el temporizador.

Por otra parte, el tiempo de adquisición y representación de los dispositivos DC, Digitales y RF es de 15 s, por consiguiente, el número de errores teóricos por dispositivo transcurrido 32 h es de 3.840. Tras realizar la simulación, se ha verificado de manera práctica el valor teórico calculado en los documentos Excel generados.

4.2. Sistema de control hardware.

De igual modo que el apartado anterior, en esta sección se detallan las pruebas realizadas para comprobar el funcionamiento del sistema hardware diseñado.

4.2.1. Prueba 1. Adquisición de datos y reinicio de dispositivos.

Descripción general

Los controladores han de ser capaces de adquirir los datos de los dispositivos de manera correcta y efectuar el reinicio de éstos en función de los requisitos de las tablas 5, 6 y 7. Con esta prueba se comprueba el diseño hardware de los controladores, la correcta programación de adquisición de datos y la programación del reinicio de dispositivos.

Procedimiento para la comprobación.

1. Realizar el montaje hardware de los controlador DC, Digital y RF e iniciar su correspondiente *sketch.*
2. Predefinir diez valores correcto e incorrecto por cada dispositivo, según la tabla 5, 6 y 10.
3. Conectar un osciloscopio a los pines de salida/*reset*. Se ha de comprobar que tienen un nivel de 5 V o 0 V en función de los datos de entrada.
4. Generar los valores predefinidos con una fuente de alimentación.
5. Comprobar simultáneamente la adquisición de datos y los pines de reinicio.

Resultados

Se comprueba la correcta adquisición de datos por parte de los controladores. Por otro lado, se verifica que, para valores de entrada incorrectos, los pines digitales de reinicio actúan correctamente.

4.2.2. Prueba 2. Transmisión de datos mediante SPI

Descripción general

Los dispositivos de control y el maestro han de ser capaces de establecer conexión y transferir datos a través del bus *SPI*. Con esta prueba se verifica la correcta implementación del protocolo diseñado y la programación realizada para la transferencia de datos.

Procedimiento para la comprobación

1. Realizar el montaje entre el controlador DC, *Arduino pro Mini,* y el maestro, *Raspberry-Pi.*
2. Inicializar el software de control para cada dispositivo.
3. Predefinir una tanta de valores que simule datos reales de los dispositivos DC.
4. Conectar el osciloscopio en los pines *SCLK* y *MOSI*.
5. Enviar datos y verificar la recepción.

Resultados

Se comprueba el correcto funcionamiento del protocolo diseñado y la correcta transferencia de datos entre los dos dispositivos. El siguiente paso consiste en verificar el protocolo de aplicación y la transferencia de datos en todo el sistema hardware de control. Para ello, se ha empleado la misma metodología con los controladores RF y Digital. El resultado ha sido satisfactorio, el maestro es capaz de coordinar correctamente a los controladores según el protocolo diseñado y la transferencia de datos entre el conjunto del sistema es correcta.

4.2.3. Prueba 3. Funcionamiento PCB.

Descripción general

El propósito de esta prueba es verificar el funcionamiento de la placa impresa diseñada. Con este objetivo, se repetirán las pruebas realizadas en los puntos anteriores.

Procedimiento para la comprobación

1. Comprobación de continuidad mediante el uso del multímetro.
2. Realizar el montaje de los controladores en la *PCB.*
3. Repetir la prueba 1 y verificar la correcta adquisición de datos y reinicio.
4. Realizar el montaje entre la *PCB* y *Raspberry-Pi.*
5. Repetir la prueba 2 y verificar la correcta transmisión de datos.

Resultados

Al realizar la prueba de continuidad, se descubre la desconexión de la pista que enlaza el pin *ground* con los pines de tierra de los dispositivos *Arduino Pro Mini.* Para solventar este inconveniente, se suelda manualmente la conexión.

Solucionado el problema, se procede a realizar la prueba 1. Se comprueba que la *PCB* funciona correctamente a la hora de adquirir datos y realizar el reinicio de dispositivos. De la misma manera, se comprueba la correcta transmisión de datos entre la *PCB* y *Raspberry-Pi.*

4.3. Funcionamiento global del sistema

Descripción general

Finalmente, en esta sección se detalla la prueba realizada para validar el sistema de control empleando todos los bloques diseñados. Con esta prueba se verifica la correcta implementación del sistema electrónico de control, la implementación de la interfaz gráfica y los protocolos diseñados para el sincronismo y la transferencia de datos entre estos dos bloques.

Procedimiento para la comprobación

1. Realizar el montaje del sistema hardware.
2. Inicializar el software de control de cada dispositivo.
3. Inicializar la interfaz gráfica.
4. Conectar *Raspberry-Pi* a la red.
5. Predefinir una tanta de valores que simule datos reales de los dispositivos DC, Digital y RF.
6. Iniciar el sistema de control por medio de la interfaz gráfica.

Resultados

Al realizar esta prueba se descubre que la interfaz gráfica es incapaz de obtener y representar los datos de los dispositivos RF y Digital, únicamente representa los valores de los dispositivos DC. Tras varias pruebas e intentos, se descubre que se ha de asignar un puerto diferente por cada *socket* creado, por tanto, se han de asignar tres puertos distintos.

Solventado el problema, se comprueba el correcto funcionamiento del sistema de control diseñado. Tras realizar esta prueba, el siguiente paso consiste en comprobar el funcionamiento con valores reales, para ello se predefinen dos tantas de valores por cada dispositivo y se generan con una fuente de alimentación. Empleando el mismo procedimiento, se comprueba que el sistema funciona correctamente.

5. Estudio económico

En este apartado se realiza un estudio de los costes del proyecto, derivados de las horas de trabajo del personal y del material empleado.

En primer lugar, se tienen en cuenta los recursos humanos, donde se ha considerado que el proyecto ha sido realizado por un ingeniero junior con un salario aproximado de 8 €/h. Para que se ajuste a la realidad, también se tienen en cuenta las horas dedicadas al aprendizaje previo de los elementos participantes en el proyecto, como el lenguaje de programación C ++, el programa *Qt* o el funcionamiento de la *Raspberry-Pi*. Así pues, se contabilizan 4 meses de trabajo, trabajando 5 horas diarias, y por tanto 400 horas dedicadas al proyecto.

En segundo lugar, se han tenido en cuenta los costes del material adquirido por el proyecto; hay que destacar que todos los entornos de programación son libres.

Finalmente, se han considerado otros recursos como son el ordenador, la licencia de Microsoft Office y el mobiliario de la zona de trabajo y herramientas. Se considera que todo el material se utiliza 6 horas al día, y por tanto unas 1.100 horas anuales. En la siguiente tabla, se muestra con mayor detalle el estudio económico desarrollado.

Concepto	Precio	Amortización	Cantidad	Coste
Recursos Materiales				
Raspberry-Pi mode b+	33,59€/u	-	1 u	33,59 €
Arduino Pro Mini	1,71€/u	-	3 u	5,13 €
LTC5533	4,55€/u	-	3 u	13,65 €
CH340G	0,70 €/u	-	1 u	0,70 €
Cable ethenet	1,4€/u	-	1 u	1,40 €
Cable USB	1,2€/u	-	1 u	1,20 €
Cable hdmi	5,5€/u	-	1 u	5,50 €
PCB	26,39€/u	-	1 u	26,39 €
Total				92,16 €
Recursos humanos				
Investigación y aprendizaje	8€/h	-	70 h	560,00 €
Elección y diseño hardware	8€/h	-	80 h	640,00 €
Diseño Sofware	8€/h	-	180 h	1.440,00 €
Redacción memoria	8€/h	-	70 h	560,00 €
Total				3.200,00 €
Otros recursos				
Sofware(Raspbian, Qt, eagle..)	0€/u	-	250h	- €
Licencia Microsoft Office	70,00 €	0,06€/h	120h	7,20 €
Ordenador	800,00 €	0,73€/h	380h	277,40 €
Herramientas y zona de trabajo	350,00 €	0,32€/h	400h	128,00 €
Total				412,60 €
TOTAL GLOBAL				3.704,76 €

Tabla 6.-Coste del prototipo diseñado. Fuente: Elaboración propia.

6. Conclusiones y desarrollo futuro

6.1. Conclusiones

Este proyecto se había marcado como objetivo diseñar un sistema de control capaz de supervisar y controlar un sistema complejo con múltiples dispositivos que generan señales de naturaleza diferente. Con esta meta, se ha desarrollado un sistema de control basado en dispositivos *SBC y* microcontroladores logrando un sistema eficiente, escalable, estructurado y con un precio reducido en comparación con otras tecnologías de control. Como resultado de la implantación se consigue el control y la automatización del sistema hardware *GEOSAR*.

Las conclusiones extraídas al realizar este proyecto son varias, destacando las numerosas ventajas y posibilidades que ofrece el uso de las plataformas *SBC* sobre otros dispositivos para aplicaciones como la desarrollada en este proyecto. El desarrollo electrónico se ha llevado a cabo mediante la combinación de Raspberry-Pi y Arduino Pro Mini, una combinación que ha sido analizada en diferentes fases del proyecto, y que ha resultado ser muy adecuada para la implementación del sistema desarrollado.

En relación con el desarrollo de la interfaz gráfica, implementada en C++, empleando el entorno *QtCreator*, aunque en un comienzo su estudio y aplicación parecía complicado y se echaba de falta algunas características muy útiles que presentan otros entornos de desarrollo, como la necesidad de crear hilos para procesar eventos, a lo largo del trabajo se han ido descubriendo de manera práctica otras propiedades interesantes, como el uso de señales para dotar de mayor inteligencia a la interfaz. El resultado de la interfaz gráfica ha sido satisfactorio. Se ha conseguido crear una interfaz intuitiva, estructurada y con un diseño atractivo, que cumple con los requisitos establecidos.

Cabe destacar como el correcto análisis del sistema, elaborando de forma satisfactoria los requisitos y casos de uso teniendo en cuenta las necesidades presentadas por el tutor ha facilitado el desarrollo del proyecto, logrando alcanzar los objetivos prefijados al inicio del proyecto.

En cuanto al aspecto personal, la realización de este proyecto ha servido para profundizar en muchos conocimientos y lecciones conseguidas a lo largo de la carrera, logrando despejar muchas dudas surgidas durante estos años y logrando alcanzar un mayor nivel en cuanto al conocimiento adquirido.

En ocasiones, también han surgido varias dificultades tales como la configuración de *Raspberry-Pi*, la instalación del entorno *QtCreator* o la creación del ejecutable de la interfaz gráfica. Finalmente han podido ser solventados sin mayores consecuencias. Esto hace que esta experiencia sea aún más enriquecedora.

6.2. Líneas futuras.

A pesar de que el sistema de control, una vez finalizado su desarrollo, cumple con los requisitos establecido, es cierto que pueden añadirse múltiples funcionalidades que puedan convertirlo en una herramienta más completa.

A continuación, se enumeran una serie de mejoras que se han ido descubriendo a lo largo del trabajo y que, por cuestiones de tiempo o presupuesto, no se han llevado a cabo.

- Controlar la potencia RF de los dispositivos. Es necesario disponer de un dispositivo que transforme la potencia de entrada en un voltaje de salida. Además, es necesario coordinar el controlar RF y el nuevo controlador para determinar la frecuencia de trabajo y posteriormente la potencia de los dispositivos RF.
- Diseñar y fabricar algún tipo de utensilio para almacenar el sistema hardware de control.
- Perfeccionar y optimizar el procesado de eventos de la interfaz gráfica.

7. Bibliografía

[1] Martin, R., Fernández, M., & Broquetas, A. (2017-06-01). Interferometric orbit determination for geostationary satellites. *Science China information Sciences*, 1-5.

[2] Josep, S. C. STD (*Sistemes de Transport de Dades*). Universitat Politécnica de Catalunya. From: *http://personals.ac.upc.es/jsunyol/STD-T1.pdf*

[3] Tomiyasu, K, (Jan 01, 1978). Synthetic aperture radar in geosynchronous orbit. *VALLEY FORGE SPACE CENTER*, 1-5.

[4] Rodon, J. R. (01, 2104). *Geosynchronous Synthetic Aperture Radar for Earth Continuous Observation Missions PhD. Universitat Politécnica de Catalunya, 29-30.*

[5] Geoscience and Remote Sensing Sociey. IEE-GRSS. From: https://www.grss-ieee.org/

[6] Vosough, V. a. (11/2011). *PLC and its Applications*. International Journal of Multidisciplinary Sciences and Engineering. VOL. 2.

[7] Murillo, P. (2015). *Efn.uncor. Introduction to PLC. Universidad Pública de Cordoba*. From: http://www.efn.uncor.edu/departamentos/electro/cat/eye_archivos/apuntes/a_practico/CAP%209%20Pco.pdf

[8] Gunter,G., Bettina,W. (02/ 2007). *Introduction to Microcontrollers*. *Vienna University of Thecnology.* From ti.tuwien.ac: https://ti.tuwien.ac.at/ecs/teaching/courses/mclu/theory-material/Microcontroller.pdf

[9] *Arduino* Pro mini Documentation (n.d). from https://store.arduino.cc/arduino-pro-mini

[10] Raspberry Pi Documentation. (n.d.). from https://www.raspberrypi.org/documentation/

[11] *IDE Arduino* Documentation (n.d). from de https://www.arduino.cc/en/main/software

[12] *Qt* Documentation (s.f.). from http://doc.qt.io/

[13] TCL5533 *Linear Technology .(*n.d) from http://cds.linear.com/docs/en/datasheet/5533f.pdf

[14] Raspberry Pi Products (n.d) from https://www.raspberrypi.org/products/raspberry-pi-1-model-b/.

[15] Enrique,A.T.(05/2002). *Sistemas básicos con Sockets.* Universidad de Málaga. From: http://www3.uji.es/~ochera/curso_2003_2004/e52/tema6-sockets.pdf